VMware GemFire 9.10 Documentation

VMware GemFire 9.10

by Broadcom

You can find the most up-to-date technical documentation on the VMware by Broadcom website at:

https://docs.vmware.com/

VMware by Broadcom
3401 Hillview Ave.
Palo Alto, CA 94304
www.vmware.com

Copyright © 2024 Broadcom. All Rights Reserved. The term “Broadcom” refers to Broadcom Inc. and/or its subsidiaries. For more information, go to https://www.broadcom.com. All trademarks, trade names, service marks, and logos referenced herein belong to their respective companies. Copyright and trademark information.

Contents

VMware GemFire® 9.10 Documentation

This documentation describes product concepts and provides complete setup instructions for VMware GemFire.

VMware GemFire shares a code base and documentation with Apache Geode. In this documentation, “VMware GemFire” and “Apache Geode” are equivalent terms.

VMware GemFire 9.10

- VMware GemFire 9.10 Release Notes
- Supported Configurations and System Requirements
- Getting Started
    - VMware GemFire in 15 Minutes or Less
    - Installing VMware GemFire
    - Upgrading VMware GemFire
- Configuring and Running a Cluster
- Basic Configuration and Programming
- Topologies and Communication
- Managing VMware GemFire
- Developing with VMware GemFire
- Developing REST Applications for VMware GemFire
- Tools and Modules
- Reference
- Experimental Features

API Reference Documentation

- VMware GemFire 9.10 Java API
- VMware GemFire Developer REST API

Documentation of Related Products

- Native Client for VMware GemFire
- Node.js Client for VMware GemFire

VMware GemFire® 9.10 Documentation

This documentation describes product concepts and provides complete setup instructions for VMware GemFire.

VMware GemFire shares a code base and documentation with Apache Geode. In this documentation, “VMware GemFire” and “Apache Geode” are equivalent terms.

VMware GemFire 9.10

- VMware GemFire 9.10 Release Notes
- Supported Configurations and System Requirements
- Getting Started
  - VMware GemFire in 15 Minutes or Less
  - Installing VMware GemFire
  - Upgrading VMware GemFire
- Configuring and Running a Cluster
- Basic Configuration and Programming
- Topologies and Communication
- Managing VMware GemFire
- Developing with VMware GemFire
- Developing REST Applications for VMware GemFire
- Tools and Modules
- Reference
- Experimental Features

API Reference Documentation

- VMware GemFire 9.10 Java API
- VMware GemFire Developer REST API

Documentation of Related Products

- Native Client for VMware GemFire
- Node.js Client for VMware GemFire

VMware GemFire 9.10 Release Notes

VMware GemFire is based on Apache Geode, and they share a common set of documentation. Therefore, references to both VMware GemFire and Apache Geode appear throughout this documentation; consider them equivalents.

NOTE: Version 9.10.14 of VMware GemFire incorporates the fix for a security issue in the log4j component. We recommend that all customers running versions prior to 9.10.14 update to the latest maintenance release. For more information on these vulnerabilities and their impact on VMware products please see VMSA-2021-0028.

Note: VMware GemFire version 9.10 requires Java 8 release 272 or a more recent version 8 update.

What's New in VMware GemFire 9.10.18

Released: June 13, 2023

VMware GemFire 9.10.18 is a maintenance release, which includes fixes for the following security issues:

- CVE-2022-1471
- CVE-2022-41881
- CVE-2022-45688
- CVE-2023-20860
- CVE-2023-22602
- CVE-2023-24998

See Issues Resolved in VMware GemFire 9.10.18 for details.

What's New in VMware GemFire 9.10.17

Released: November 3, 2022

VMware GemFire 9.10.17 is a maintenance release, which includes fixes for the following security issues:

- CVE-2022-3171
- CVE-2022-40664
- CVE-2022-42003

- CVE-2022-42889

See Issues Resolved in VMware GemFire 9.10.17 for details.

What’s New in VMware GemFire 9.10.16

VMware GemFire 9.10.16 is a maintenance release, which includes fixes for the following security issues:

- CVE-2022-32532
- CVE-2022-25647
- CVE-2022-22978
- CVE-2022-22968
- CVE-2020-36518
- CVE-2019-17495
- CVE-2016-1000027

See Issues Resolved in VMware GemFire 9.10.16 for details.

What’s New in VMware GemFire 9.10.15

VMware GemFire 9.10.15 is based on Apache Geode 1.12.9.

VMware GemFire 9.10.15 includes a fix for the following security issues:

- CVE-2022-23207
- CVE-2022-34870

The VMware GemFire 9.10.15 release includes the following improvement:

GEM-3518: Prometheus metrics, formerly a single JAR file (`prometheus-metrics.jar`), are now distributed as a directory containing multiple JARs (`gemfire-prometheus-metrics/*.jar`). If you load the Prometheus JAR explicitly, in a gfsh command for example, you must change the syntax of the path to load the JARs from the directory.

For example, if your product distribution is located in `/gemfire`, replace the single-JAR specification

```
/gemfire/tools/Modules/prometheus-metrics.jar
```

with

```
/gemfire/tools/Modules/gemfire-prometheus-metrics/*.jar
```

In the gfsh `start locator` or `start server` command, this would appear as:

```
--classpath=/gemfire/tools/Modules/gemfire-prometheus-metrics/*.jar
```

See Issues Resolved in VMware GemFire 9.10.15 for details regarding issues addressed in this release.

What’s New in VMware GemFire 9.10.14

VMware GemFire 9.10.14 is based on Apache Geode 1.12.8.

VMware GemFire 9.10.14 is a maintenance release. See Issues Resolved in VMware GemFire 9.10.14 for details regarding issues addressed in this release.

GEODE-9905, GEM-3490: Upgraded log4j to v2.17.1 to address CVE-2021-44832 and CVE-2021-45105.

What’s New in VMware GemFire 9.10.13

VMware GemFire 9.10.13 is based on Apache Geode 1.12.7.

VMware GemFire 9.10.13 is a maintenance release.

GEODE-9898, GEM-3486: Upgraded log4j to v2.16.0 to address CVE-2021-45046.

What’s New in VMware GemFire 9.10.12

VMware GemFire 9.10.12 is based on Apache Geode 1.12.6.

VMware GemFire 9.10.12 is a maintenance release.

GEODE-9888, GEM-3478: Upgraded log4j to v2.15.0 to address CVE-2021-44228.

What’s New in VMware GemFire 9.10.11

VMware GemFire 9.10.11 is based on Apache Geode 1.12.5.

VMware GemFire 9.10.11 is a maintenance release. See Issues Resolved in VMware GemFire 9.10.11 for details regarding issues addressed in this release.

New in this release:

- VMware GemFire no longer supports Homebrew installation.

What’s New in VMware GemFire 9.10.10

VMware GemFire 9.10.10 is based on Apache Geode 1.12.5.

VMware GemFire 9.10.10 is a maintenance release. See Issues Resolved in VMware GemFire 9.10.10 for details regarding issues addressed in this release.

What’s New in VMware GemFire 9.10.9

VMware GemFire 9.10.9 is based on Apache Geode 1.12.4.

VMware GemFire 9.10.9 includes the following new features and improvements:

- VMware GemFire 9.10.9 provides a new option enabling automatic retries in response to PdxSerializationException. To apply this mode, the client application must set the system property `gemfire.enableQueryRetryOnPdxSerializationException=true`.

  For example:

```
client.invoke(() -> {
  System.setProperty(
    GeodeGlossary.GEMFIRE_PREFIX + "enableQueryRetryOnPdxSerializationExceptio
n", "true");
  QueryService remoteQueryService = (PoolManager.find(poolName)).getQueryServic
e();
  Query query = remoteQueryService.newQuery("SELECT DISTINCT id FROM pdxTest");
  SelectResults<TestObjectThrowsPdxSerializationException> selectResults = (Sel
ectResults)query.execute();
});
```

What's New in VMware GemFire 9.10.8

VMware GemFire 9.10.8 is based on Apache Geode 1.12.3.

VMware GemFire 9.10.8 includes the following new features and improvements:

- VMware GemFire version 9.10.8 introduces a VMware GemFire metrics module that provides out-of-the-box integration with Tanzu Observability by Wavefront. Metrics can be forwarded to Wavefront for analysis and alerting. For more information, see Tanzu Observability by Wavefront.
- Documentation was added for VMware GemFire Micrometer, an interface for emitting user-focused metrics to a variety of different consuming Application Performance Monitoring (APM) tools. This feature was released in Apache Geode v1.11, and so was included in the Tanzu Gemfire 9.10 release, but it was not previously documented in the VMware GemFire User Guide.
- Addressed an issue with deserialization of configuration information from old locators during rolling upgrades (**GEODE-9289, GEM-3253**).
- A new default setting for a JMX property affecting users of the Java 11 JDK (**GEODE-9064, GEM-3185**).

What's New in VMware GemFire 9.10.7

VMware GemFire 9.10.7 is based on Apache Geode 1.12.2.

VMware GemFire versions 9.10.1 through 9.10.7 are maintenance releases that resolve a variety of issues—see Resolved Issues for more information.

What's New in VMware GemFire 9.10

VMware GemFire 9.10 includes the following improvements:

- VMware GemFire 9.10 is based on Apache Geode version 1.12, so it also includes features and bug fixes from Apache Geode version 1.11.
- JMX now exposes the free disk space percentage as a statistic.
- Asynchronous event queues may now be created in a paused state. This allows better control of event dispatch, as the dispatch may be delayed until the application is ready for event consumption.

- The new pluggable OQL security framework restrict which classes and methods can be called from queries. A system administrator can change the security posture at runtime. Provided implementations cover several use cases, as described in Method Invocation Authorizers.
- There are new endpoints in the REST API for Management, including create and delete index, list deployed jars, show PDX configuration, and show the supported REST API versions.

Note: VMware GemFire version 9.10 requires Java 8 release 272 or a more recent version 8 update.

Installing VMware GemFire 9.10

You can download VMware GemFire 9.10 from the VMware GemFire product download page.

For details on how to install VMware GemFire 9.10, see Installing VMware GemFire.

Upgrading to VMware GemFire 9.10

To upgrade from an earlier version of VMware GemFire to the current version, see Upgrading VMware GemFire. Version-specific upgrade details follow.

Upgrading to Version 9.10.6

The fix for **issues GEODE-8799 and GEM-3041:** addressed a performance issue by increasing the defaults for maximum number of pooled message processor threads and maximum partitioned region message processor threads. NOTE: Users should be aware that if their system had been hitting the old, lower default maximums, then upgrading may cause increased use of system resources as the system is no longer constrained to the old values. If your system depends on constraining system resources to the old default values, you can set these properties explicitly using DistributionManager.MAX_THREADS and DistributionManager.MAX_PR_THREADS, respectively.

The following table shows the old and new maximum values.

System Resource	Old Default Value	New Default Value
DistributionManager.MAX_THREADS	100	1000
DistributionManager.MAX_PR_THREADS	The greater of (CPUs * 4) or 16	The greater of (CPUs * 32) or 200

Upgrading from a Version prior to 9.1.1

When servers are upgraded to this VMware GemFire version from a version prior to version 9.1.1, their authentication of client messages is deactivated to facilitate rolling upgrades. If your clients are configured to send authenticated messages, the servers will honor those messages, but will not enforce authentication until you reinstate authentication on each of the upgraded servers.

To reinstate secure client/server messaging, restart each server with the `geode.disallow-internal-messages-without-credentials` system property set to `true`. For example:

```
gfsh>start server --name=server_name --dir=server_config_dir \
--J=-Dgeode.disallow-internal-messages-without-credentials=true
```

Resolved Issues

The VMware GemFire 9.10 release is based on the Apache Geode 1.12 release, which also includes bug fixes from the Apache Geode 1.11 release. This section describes issue resolutions that significantly affect VMware GemFire applications. For a list of all issues resolved in Apache Geode 1.12, see the [Geode 1.12 Release Notes] (https://cwiki.apache.org/confluence/display/GEODE/Release+Notes#ReleaseNotes-1.12.0). For a list of all issues resolved in Apache Geode 1.11, see the [Geode 1.11 Release Notes] (https://cwiki.apache.org/confluence/display/GEODE/Release+Notes#ReleaseNotes-1.11.0). Ticket numbers of the form GEODE-NNNN can be inspected at the Apache Software Foundation site for Geode issues.

Issues Resolved in VMware GemFire 9.10.18

GEM-4717: Fixed an issue with session MaxInactiveTime. Setting a new session inactive time now supersedes the old value as expected.

GEM-4864: Allow an older gfsh client to connect to a newer cluster across major versions.

GEM-5053: Updated netty-transport to 4.1.86 to address CVE-2022-41881.

GEM-5144: Updated shiro-core to 1.11.0 to address CVE-2023-22602.

GEM-5253: Updated commons-fileupload to 1.5 address CVE-2023-24998.

GEM-5318: Updated snakeyaml to 2.0 to address CVE-2022-1471.

GEM-5335: Updated spring_framework to 5.3.26 to address CVE-2023-20860.

GEM-4096: In Pulse, fixed StatRate to correctly calculate the rate per specified period.

GEM-4915: Improved tombstone expiration time calculations to consider large clock jumps.

Issues Resolved in VMware GemFire 9.10.17

GEM-2230: Fixed a hang that occurred during initialization in fixed partitioning when the configured primary bucket is still being initialized.

GEM-3103: Fixed a hang that could occur when rebalancing a partioned region.

GEM-3168, GEM-3729: Fixed an issue that prevented recovery following a network partition.

GEM-4331: Implemented wider protections against cross-site scripting (XSS) attacks.

GEM-4647: Updated FasterXML Jackson Databind to version 2.13.4.2. This addresses CVE-2022-42003.

GEM-4746: Updated protobuf-java to version 3.21.8. This addresses CVE-2022-3171.

GEM-4703: Updated Apache Shiro to version 1.10.0. This addresses CVE-2022-40664.

GEM-4710: Updated Apache commons-text to version 1.10.0. This addresses CVE-2022-42889.

Issues Resolved in VMware GemFire 9.10.16

GEM-3036, GEM-3507, GEM-3691: Fixed a performance issue caused by a memberId leak in non-persistent regions.

GEM-3214: Introduce a performance improvement by increasing the default BridgeServer.HANDSHAKE_POOL_SIZE from 4 to 50.

GEM-3342: Fixed a problem with stuck function threads in long-running TSL 1.3 sessions.

GEM-3433: Restored the visibility of processCpuTime statistics hidden by Java 11.

GEM-3549: Removed an unnecessary restriction on subregion rebalancing. The –include-region option is no longer mandatory.

GEM-3567: Upgraded jackson to v2.13.2.2 to address CVE-2020-36518.

GEM-3580, GEM-3627: Upgraded spring to v5.3.20 to address CVE-2016-1000027 and CVE-2022-22968.

GEM-3581: Upgraded springdoc to v2.2.0 and classgraph to v4.8.146 to address CVE-2019-17495.

GEM-3647: Upgraded gson to v2.8.9 to address CVE-2022-25647.

GEM-3677: Upgraded spring-security to v5.5.8, commons-lang3 to v3.12.0, and slf4j to v1.7.32 to address CVE-2022-22978.

GEM-3684: Removed Server: header from all HTTP responses to mitigate a potential security risk in the REST API.

GEM-3722: Corrected a cross-site scripting (XSS) vulnerability in Pulse.

GEM-3749: Each GemfireHttpSession is now sized, instead of only the first instance being sized.

GEM-3750: To improve size estimation, modified the ObjectSizer to not size Thread and ThreadGroup instances by default.

GEM-3761: Fixed ExtendedNumericComparator to properly handle NULL and UNDEFINED comparison.

GEM-3766: Fixed a case in which findDistributedMembers() throws an unsupported operation exception.

GEM-3773: Upgraded shiro to v1.9.1 to address CVE-2022-32532.

GEODE-9910: Fixed a problem in which nodes with embedded locators failed to auto-reconnect following a network partition.

Issues Resolved in VMware GemFire 9.10.15

GEODE-9819: Repaired a server socket leak. The leak happened only when using durable clients. If a durable client failed to connect with ServerRefusedConnectionException, then a socket was leaked on the server.

GEODE-9372: Added createSenderTime and createSenderInProgress stats to DistributionStats to help diagnose data replication spikes.

GEODE-10093, GEM-3541: Fixed an issue in which the DeltaSession getAttribute method logs an NPE and returns an unserialized value when called on an attribute with a null value.

Issues Resolved in VMware GemFire 9.10.14

GEODE-9060, GEM-3488 Cleaned up replicates list for the GII (Get Initial Image) provider to improve restarts.

GEODE-9905, GEM-3490: Upgraded log4j to v2.17.1 to address CVE-2021-44832 and CVE-2021-45105.

Issues Resolved in VMware GemFire 9.10.13

GEODE-9898, GEM-3486: Upgraded log4j to v2.16.0 to address CVE-2021-45046.

Issues Resolved in VMware GemFire 9.10.12

GEODE-9888, GEM-3478: Upgraded log4j to address CVE-2021-44228.

Issues Resolved in VMware GemFire 9.10.11

GEODE-7920, GEM-2863: Improved responsiveness of membership messaging by disallowing the processing of cache operations directly on the limited pool of membership messaging threads.

GEODE-8542: Limited the size of message chunks to the maximum message size allowed by `org.apache.geode.internal.tcp.Connection`.

GEODE-9714, GEODE-9486: Fixed a problem in which some serialized classes failed to deserialize when `validate-serializable-objects` is enabled.

GEODE-9767, GEM-3423: Upgraded netty to address CVE-2021-37136 and CVE-2021-37137.

GEODE-9783, GEM-3422: Removed unnecessary `.jar` files from `pulse.war`.

GEODE-9825, GEM-3444: Fixed network buffer handling problem (when TLS was deactivated, and cluster members used disparate `socket-buffer-size settings`) that could result in hangs.

GEODE-9838, GEM-3445: Improved index maintenance and reliability by adding key details to debug-level index update logs when an operation fails.

Issues Resolved in VMware GemFire 9.10.10

GEODE-9486: Fixed a case in which serialized classes failed to deserialize when validate-serializable-objects was enabled.

GEODE-9515: Fixed a situation in which JMX manager failed to start because initialization of an MBeanServer object was attempted twice.

GEODE-9554, GEM-3345: Fixed a case in which rebalancing a region with multiple redundancy zones could fail due to the deletion of a bucket in the wrong redundancy zone.

GEODE-9578, GEM-3362: Upgraded spring-security to address CVE-2021-22119.

GEODE-9596, GEM-3326: Fixed an issue in which continuous query events could be lost due to an HAContainer's eviction policy.

GEODE-9640, GEM-3373: Fixed a case in which cluster restart issued duplicate event IDs, causing new operations to be lost on the client.

GEODE-9655, GEM-3245: Updated the Shiro component to take advantage of its improved parsing behavior.

Issues Resolved in VMware GemFire 9.10.9

GEODE-9141, GEM-3219: Fixed a corrupted DestroyRegionMessage that could cause a cache server to hang during shutdown.

GEODE-9180: Improved diagnostic logging by adding a warning when a heartbeat generation thread oversleeps.

GEODE-9295, GEM-3310: Corrected an issue with entry idle expiration that could deactivate all expiration, resulting in symptoms such as a drastic increase in session state region entry counts following an upgrade.

GEODE-9346, GEM-2816: Introduced an option allowing a client application to enable automatic retries in response to PdxSerializationException by setting the system property `gemfire.enableQueryRetryOnPdxSerializationException=true`.

GEODE-9380, GEM-3301: Addressed a significant performance degradation in peer-to-peer TLS handshake times by replacing sleep() calls with thread yields.

Issues Resolved in VMware GemFire 9.10.8

GEODE-9064, GEM-3185: Serialization filtering for JMX/RMI is configured by default on Java 11.

GEODE-9146, GEM-3223: Entry idle expiration on regions with eviction policy set to “destroy” will now expire entries consistently across the cluster.

GEODE-9289, GEM-3253: Addressed an issue with deserialization of configuration information from old locators during rolling upgrades.

GEODE-9307, GEM-3293: Corrected increased heap consumption following auto-reconnection caused by region references that should have been destroyed following a forced disconnect.

GEODE-9331, GEM-3300: Improved the efficiency of creating peer-to-peer connections when `conserve-sockets=false` by eliminating a redundant list of weak references.

GEODE-9339, GEM-3302: Upgraded json-smart to address CVE-2021-27568.

GEODE-9363, GEM-3306: Upgraded spring-core to address CVE-2021-22118.

Issues Resolved in VMware GemFire 9.10.7

GEODE-8217, GEM-2909: Fixed a problem with session state serialization and deserialization.

GEODE-8221, GEM-2908: Fixed a failure to update session state.

GEODE-8513, GEM-3030: Cleaned up inefficiencies and spurious errors when storing and retrieving session state information.

GEODE-8558, GEM-3077: Fixed syntax issue in the syntax for commented Pulse queries.

GEODE-8623, GEM-3088: Corrected a timing issue between DNS and Geode startup that could result in permanent unknown host exceptions.

GEODE-8671, GEM-3095: Fixed a serialization data corruption issue that arose when two threads simultaneously accessed a single PdxInstance.

GEODE-8684, GEM-3254: Ensured that the `maxInactiveInterval` setting is honored when the commit valve is deactivated.

GEODE-8781, GEM- 3252: Fixed a problem with upgrades when attempting to restart from API-configured regions.

GEODE-8815, GEM-3128: Fixed a case in which cache closure due to an uncaught exception during member startup was incorrectly treated as a graceful shutdown.

GEODE-8926, GEM-3184: Fixed a problem with missed Continuous Query events when they occurred at the same time a post-CQ function was executing.

GEODE-8958, GEM-3027: Improved tombstone expiration logic with regard to the handling of future timestamps.

GEODE-8974, GEM-3205: Fixed vulnerability CVE-2020-13956 in `apache-httpclient`.

GEODE-8989, GEM-3211: Fixed vulnerability CVE-2021-22112 in `spring-security`.

GEODE-8996, GEM-3158: Re-established backward compatibility for rebalance and restore redundancy commands.

GEODE-9010, GEM-3218: Fixed vulnerability CVE-2020-27223 in `jetty`.

GEODE-9016, GEM-3212: Fixed a null-pointer exception that could occur when a continuous query generated a local-destroy event.

GEODE-9030, GEM-3221: Fixed a spurious Region Destroyed exception that could occur when a query was initiated on a partitioned region immediately following a startup or rebalance operation.

GEODE-9040, GEM-3215: Improved the handling of missing colocated regions, so that the `SingleThreadColocationLogger` now stops, as it should.

GEODE-9051: Added a feature to measure tenured heap consumption and record the information in the logs after garbage collection.

GEODE-9126, GEM-3251: Fixed vulnerability CVE-2021-28165 and CVE-2021-28164 in `jetty`.

Issues Resolved in VMware GemFire 9.10.6

GEODE-2644, GEM-3138: Restored member name to log entries.

GEODE-5922, GEM-3155: Fixed a performance issue in the event queues of WAN-connected clusters.

GEODE-7884, GEM-1535: Fixed a condition in which a server would hang during a cache close operation with an IllegalStateException error, due to a timer being set on an already-completed operation.

GEODE-8261, GEM-2975, GEM-3130: Fixed a null-pointer error that could occur when a client attempts to register interest after server shutdown has been initiated.

GEODE-8419, GEM-3079: SSL/TLS protocol, cipher suite configurations and per-component TLS properties were being ignored in some cases; these settings are now respected.

GEODE-8447, GEM-3019: The output of localized dates in log timestamps and Pulse query results now always include seconds. A previous fix for GEODE-8447 revised Pulse to display dates using the local time zone.

GEODE-8536, GEM-2872, GEM-2961, GEM-2997: Fixed a stack overflow that could occur when Lucene IndexWriter was unable to be created.

GEODE-8685, GEM-3174: Fixed export feature to no longer deserialize region values and classes.

GEODE-8686, GEM-1681: Dispensed with a rarely-called tombstone removal optimization that could occasionally cause a deadlock that prevented the completion of region creation.

GEODE-8721, GEM-3110: Fixed a condition under which the losing side of a network partition failed to shut down.

GEODE-8734, GEM-3104: Implemented a more robust approach for generating Geode statistics that accommodates a variety of `/proc/net/netstat` formats produced by different versions of Linux.

GEODE-8764, GEM-3148: Fixed an authentication problem that erroneously blocked the execution of Lucene queries requiring region-level permissions.

GEODE-8779, GEM-3143: Eliminated a case in which session management messages were sent unnecessarily to clients whose local caches were not enabled.

GEODE-8782: Added the ability to retrieve the Principal from the FunctionContext when a SecurityManager is enabled.

GEODE-8795, GEM-3111: Lucene queries now utilize security post-processing, if enabled.

GEODE-8799, GEM-3041: Addressed a performance issue by increasing the defaults for maximum number of pooled message processor threads and maximum partitioned region message processor threads. If your system depends on constraining system resources to the old default values, you can set these properties explicitly using DistributionManager.MAX_THREADS and DistributionManager.MAX_PR_THREADS, respectively.

The following table shows the old and new maximum values.

System Resource	Old Default Value	New Default Value
DistributionManager.MAX_THREADS	100	1000
DistributionManager.MAX_PR_THREADS	The greater of (CPUs * 4) or 16	The greater of (CPUs * 32) or 200

GEODE-8895, GEM-3170: Handle an unexpected socket closure with more grace by implementing retries internally, such that the client no longer needs to deal with an `InternalGemFireException`.

GEODE-8930, GEM-2677: Fixed a race condition that caused a deadlock when executing a create operation within a transaction with `conserve-sockets` set to `true`. With this fix, the deadlock no longer occurs, but best practice is to set `conserve-sockets` to `false` when using transactions.

Issues Resolved in VMware GemFire 9.10.5

GEODE-8238, GEM-2901: Improved connection close behavior to ensure final delivery of cluster messages during shutdown and avoid potential hangs.

GEODE-8432, GEM-2778, GEM-3026, GEM-3029: Fixed a hang that occurred with asynchronous event queues during rebalance operations.

GEODE-8475, GEM-1589: Resolved a deadlock in ParallelGatewaySenderQueue.

GEODE-8520, GEM-3056: GarbageCollectionCount metric no longer shows negative values. GCStatsMonitor now sums up all the GC stats to get the total GC count and GC time.

GEODE-8557, GEM-3066, GEM-3067: Reclassified “java.lang.IllegalStateException: NioSslEngine has been closed” to IOException, which allows retries of server connection failures.

GEODE-8564, GEM-3081: Fixed an exception thrown by an attempt to remove a managed connection that has already been invalidated. Corrected functionality to avoid unnecessary intermediary collection creations.

GEODE-8584, GEM-3092: Corrected a peer-to-peer messaging failure with TLS when attempting to replicate data while a connection is being closed.

GEODE-8651, GEM-3109: Corrected a problem that caused TLS-encrypted messaging to hang when conserve-sockets is set to false.

GEODE-8652, GEM-3099: Corrected a synchronization issue that caused TLS-encrypted message transmission to hang during shutdown.

Issues Resolved in VMware GemFire 9.10.4

GEODE-8385, GEM-2936: Fixed a problem with cluster shutdown which caused a hang on restart, due to ambiguity as to which node had the most up-to-date disk store. This could occur if nodes (locators or servers) were simultaneously shut down.

GEODE-8463, GEODE-8506, GEM-3020: Refined the behavior of BufferPool to always return a buffer that has exactly the requested capacity. In the past, BufferPool could return a buffer larger than the size requested, which could lead to decryption errors and lost messages when using TLS(SSL) protocol TLSv1.3.

GEODE-8478, GEM-3069: Fixed an issue that caused a gateway sender to shut down if (1) its alert-threshold was configured, (2) an event was on the queue longer than that specified threshold, (3) the logger was attempting to record an alert for that message, and (4) a field in the event object threw an exception while undergoing `toString` conversion.

GEODE-8489, GEM-3043: Restored the behavior of Pulse queries for consistency with earlier versions. For example, the result of a `select *` query once again includes attributes without values.

Issues Resolved in VMware GemFire 9.10.3

GEODE-6564, GEM-3001: Fixed a memory leak that occurred when a replicated region, configured with entry expiration, was cleared.

GEODE-8331: Fixed an issue that prevented gfsh v1.12 (GemFire 9.10) from connecting to a Geode v1.10 (GemFire 9.9) server. Now, if a Geode v1.10 (GemFire 9.9) command is incompatible with a command issued by the newer version of gfsh, it will fail with a detailed error message.

GEODE-8394, GEM-2989: Fixed an issue in which putAll operations or put operations with large objects could result in data corruption if invoked without having a sufficiently large read-timeout. Subsequent get operations or queries could experience failures due to this corruption.

GEODE-8447, GEM-3019: Pulse displays dates using the local time zone.

GEODE-8483, GEM-3004: Introduced a correction to JCA transaction logic that restores the ability of the system to detect commit conflict exceptions in concurrent access situations.

Issues Resolved in VMware GemFire 9.10.2

GEODE-8174, GEM-2884: Fixed an issue that caused an incorrect ConcurrentModificationException to be thrown when using JTA transactions.

- **GEODE-8029, GEM-2919:** Unused disk store backups (`.drf` files) are now deleted to prevent the proliferation of unused records and files, which could cause members to fail during startup while recovering disk stores.

GEODE-8131, GEM-2938: Eliminated a point of contention in the alert logging system that could occasionally block cache operations.

GEODE-8195: Improved WAN retry logic.

GEODE-8240: After a rolling upgrade, even though all locators were upgraded, the membership view indicated some were running the old version. This would have been evident in the gfsh list members command and in logs. The correct version number is now reported.

GEODE-8259, GEM-2943: When a client performs a single-hop `getAll()` operation and encounters a serialization error, the operation is now re-tried.

GEODE-8287, GEM-2897, GEM-2950: Ironed out a recently-introduced problem that caused a degradation in performance for join queries or queries that use multiple indexes. Performance in such cases has been restored to v8.2 levels.

Issues Resolved in VMware GemFire 9.10.1

GEODE-7851: Pulse logging has been restored.

GEODE-7940: Fixed a problem in which a parallel gateway sender stopped sending events if another gateway sender attached to the same region was destroyed.

GEODE-8055, GEM-2890: Indexes can now be created on subregions.

GEODE-8071, GEM-2896: Fixed the `gfsh rebalance` command to prevent locators from hanging during shutdown.

GEODE-8091, GEM-2898: Locators failed to start after upgrade to v9.10 if a deprecated gfsh `start locator` option, `--load-cluster-configuration-from-dir`, was in use. Support for the deprecated option has been restored to allow startup to proceed. Beginning with GemFire 9.5 and Geode 1.6, the gfsh command `import cluster-configuration` supersedes `start locator --load-cluster-configuration-from-dir`.

Issues Resolved in VMware GemFire 9.10

GEODE-7465, GEM-2773: Fixed an issue that threw `RegionDestroyedException` when an asynchronous event queue was stopped and then restarted.

GEODE-7473, GEM-2774: Fixed a memory leak in the accumulation of entry event objects that could occur when a gateway receiver existed for the same region as a gateway sender.

GEODE-7593, GEM-2795: Fixed a memory leak that caused a higher than expected heap size when eviction was enabled. The issue was due to indexes that retained references to serialized values when the indexed key was a PDX string.

GEODE-7374, GEM-2748: VMware GemFire no longer throws a `ClassCastException` when using the deprecated `MemberCommandService`.

GEODE-7373, GEM-2746: Enforce JMX credentials to be either a string or an array of strings.

GEODE-7036, GEM-2723: Fixed a bug that could result in a `NullPointerException` when restarting members and a locator rejoins the cluster to form a quorum.

GEODE-7264, GEM-2706: Updated the Jackson dependency to version 2.10.0.

GEODE-7334, GEM-2705: VMware GemFire no longer throws a `ClassCastException` when the developer REST API is enabled and `JodaModule` is in the classpath.

GEODE-7208, GEM-2698: Fixed Lucene index creation when there are non-primitive fields in a child class, and the fields are defined in the parent class.

GEODE-7268, GEM-2692: A `gfsh alter region` no longer causes a soft hang when many gateway senders exist, by making the gfsh command implementation work more like the equivalent Java API.

GEODE-7535, GEM-2654: Fixed a race condition that could improperly result in an `EntryDestroyedException` during a `getAll` operation, if a client proxy region exists and there are no transactions.

GEODE-7085, GEM-2647: Eliminated an `IndexOutOfBoundsException` while recovering values from disk when the region version value overflowed.

GEODE-7015, GEM-2604: Fixed a bug that could result in a hung server if a move bucket operation failed during rebalance due to a forced disconnect with recreated persistent partitioned regions.

GEODE-7042: The server launcher now waits until all of the server's startup tasks have completed before updating the server's status to "online". The `ServerLauncher.start()` method now blocks until redundancy recovery and recovery of values from disk have completed. This change has the effect of making the `gfsh start server` command wait for all server startup tasks to complete before returning.

GEODE-4993: GatewaySender connection statistics are now stored after being captured.

GEODE-6973: Improved creation time performance when there are a large number of PDX types.

GEODE-7628, GEM-2747: Block JMX MBean creation when no `SecurityManager` interface is implemented.

GEODE-7755, GEM-2614: Fixed a bug that resulted in locator processes continuing to run even when the logs show them shut down.

GEODE-7728, GEM-2819: Fixed an exception thrown when executing an equi-join query and both fields are indexed.

GEODE-7294: Moved to Spring 5 and updated other third-party libraries to their latest security patch versions.

GEODE-7310: Fixed an issue in which taking a backup was not properly aborted if a member of the distributed system was lost during the backup process.

GEODE-7750, GEODE-7760, GEODE-7796, GEM-2821, GEM-2823: Fixed an issue which could cause a locator to fail to restart properly, and also corrupted the cluster configuration.

GEODE-7763: Reduced the performance degradation caused when a client has multiple threads, and each client thread does operations on the same region key.

Support

General support includes security vulnerability resolutions and critical bug fixes in all supported minor versions, while other maintenance is applied only to the latest supported minor release.

Obtaining and Installing Security Updates

New versions of VMware GemFire often include important security fixes, so VMware recommends you keep up to date with the latest releases.

For details about any security fixes in a particular release, see the Application Security Team page.

Supported Configurations and System Requirements

The sections that follow document supported operating system platforms and describe additional system requirements for VMware GemFire.

- **VMware GemFire Supported Configurations**

  VMware GemFire is supported on a variety of platforms.

- **Host Machine Requirements**

  Each machine must meet a set of system requirements.

- **Supported Platforms and System Requirements for Tools**:

  Tools, including Pulse and VSD, are supported on a variety of platforms.

  - **Pulse System Requirements**

    Verify that your system meets the installation and runtime requirements for Pulse.

  - **VSD System Requirements**

    View a list of platforms that are known to work with VSD.

Tanzu GemFire Supported Configurations

VMware GemFire is supported on a variety of platforms.

Note: Running VMware GemFire clusters with a mix of different platforms has not been tested. We recommend that you use a consistent platform on all machines in your cluster.

Supported Platforms

All platforms listed as deprecated may be removed in a future release of the product.

These VMware GemFire 9.10 platforms provide support for production systems:

- CentOS 7 for the x86-64 processor
- RHEL 8 for the x86-64 processor
- RHEL 7 for the x86-64 processor
- RHEL 6 for the x86-64 processor **(Deprecated)**
- Ubuntu 14.04 for the x86-64 processor
- Ubuntu 16.04 for the x86-64 processor
- SUSE Linux Enterprise Server 11 for the x86-64 processor

- Solaris 11 for the x86-64 processor
- Solaris 11 for a 64-bit SPARC processor
- Windows 2012 Server R2 for the x86-64 processor
- Windows 2012 Server for the x86-64 processor **(Deprecated)**
- AIX 7 with IBM SDK, Java Technology Edition, Version 8*. This platform is only supported for GemFire application clients. The GemFire cluster must be configured on one of the other supported platforms.

These cloud platforms are supported for VMware GemFire 9.10:

- Linux platforms under AWS
- Linux platforms under Microsoft Azure

These VMware GemFire 9.10 platforms provide support for development systems:

- MacOS 10.12 (Sierra) for the x86-64 processor
- Windows 10 for the x86-64 processor

Java Support Notes

This version of VMware GemFire requires Java 8 release 272 or a more recent version 8 update. The same versions are supported with OpenJDK (HotSpot).

VMware GemFire is compatible with Open JDK 11.

The VMware GemFire product download does not include Java. Download and install a supported JRE or JDK on each system running GemFire. VMware recommends the installation of a full JDK (and not just a JRE) to obtain better performance with `gfsh status` and `gfsh stop` commands.

The IBM SDK, Java Technology Edition, Version 8 is supported for application clients only. Some client region eviction configurations such as `HEAP_LRU` are known to not function properly in this release.

File System Type for Linux Platforms

For optimal disk-store performance, VMware recommends avoiding the use of `ext3` file systems when operating on Linux platforms.

Host Machine Requirements

Each machine that will run VMware GemFire must meet the following requirements:

- Java SE Development Kit 8 with update 272 or a more recent version 8 update. The same versions are supported with OpenJDK. VMware GemFire is compatible with Open JDK 11 (HotSpot).
- An adequate per-user limit on the number of file descriptors; for Unix/Linux, the recommended soft limit is 8192, and the hard limit is 81920.
- An adequate per-user limit on the number of processes (nproc); for Unix/Linux, the recommended soft limit is 501408, with an unlimited hard limit.

- TCP/IP.
- A system clock set to the correct time and a time synchronization service such as Network Time Protocol (NTP). Correct time stamps permit the following activities:
  - Logs that are useful for troubleshooting. Synchronized time stamps ensure that log messages from different hosts can be merged to reproduce an accurate chronological history of a distributed run.
  - Aggregate product-level and application-level time statistics.
  - Accurate monitoring of the Geode system with scripts and other tools that read the system statistics and log files.
- The host name and host files are properly configured for the machine. The host name and host file configuration can affect `gfsh` and Pulse functionality.
- Deactivate TCP SYN cookies. Most default Linux installations use SYN cookies to protect the system against malicious attacks that flood TCP SYN packets, but this feature is not compatible with stable and busy VMware GemFire clusters. Security implementations should instead seek to prevent attacks by placing VMware GemFire server clusters behind advanced firewall protection.

  To deactivate SYN cookies permanently: 1. Edit the `/etc/sysctl.conf` file to include the following line:

  ~~~~
  ``` pre
  net.ipv4.tcp_syncookies = 0
  ```
  Setting this value to zero deactivates SYN cookies.
  ~~~~

  1. Reload `sysctl.conf`:

     ```
     sysctl -p
     ```

Getting Started with VMware GemFire

A tutorial demonstrates features, and a main features section describes key functionality.

- **About VMware GemFire**

  VMware GemFire is a data management platform that provides real-time, consistent access to data-intensive applications throughout widely distributed cloud architectures.

- **Main Features**

  This section summarizes the main features and key functionality.

- **Installing VMware GemFire**

- **Upgrading VMware GemFire**

- **VMware GemFire in 15 Minutes or Less**

  Take this brief tour to try out basic features and functionality.

About VMware GemFire

VMware GemFire is a data management platform that provides real-time, consistent access to data-intensive applications throughout widely distributed cloud architectures.

VMware GemFire pools memory, CPU, network resources, and optionally local disk across multiple processes to manage application objects and behavior. It uses dynamic replication and data partitioning techniques to implement high availability, improved performance, scalability, and fault tolerance. In addition to being a distributed data container, VMware GemFire is an in-memory data management system that provides reliable asynchronous event notifications and guaranteed message delivery.

Main Concepts and Components

Caches are an abstraction that describe a node in a VMware GemFire distributed system. Application architects can arrange these nodes in peer-to-peer or client/server topologies.

Within each cache, you define data *regions*. Data regions are analogous to tables in a relational database and manage data in a distributed fashion as name/value pairs. A *replicated* region stores identical copies of the data on each cache member of a distributed system. A *partitioned* region spreads the data among cache members. After the system is configured, client applications can access the distributed data in regions without knowledge of the underlying system architecture. You can define listeners to create notifications about when data has changed, and you can define expiration criteria to delete obsolete data from a region.

VMware GemFire *locators* provide both member discovery and load-balancing services. You configure clients with a list of locator services and the locators maintain a dynamic list of member servers.

VMware GemFire uses *continuous querying* to enable event-driven architectures. VMware GemFire ties events and data together so that when an event is processed, the data required to process the event is available without additional queries to a disk-based database. Clients can subscribe to change notifications so that they can execute tasks when a specific piece of data changes.

In addition to peer-to-peer and client/server topologies, VMware GemFire supports multi-site configurations that allow you to scale horizontally between disparate, loosely-coupled distributed systems over a wide geographically separated network. A wide-area network (WAN) is the main use case for the multi-site topology.

Main Features

This section summarizes main features and key functionality.

- High Read-and-Write Throughput
- Low and Predictable Latency
- High Scalability
- Continuous Availability
- Reliable Event Notifications
- Parallelized Application Behavior on Data Stores
- Shared-Nothing Disk Persistence
- Reduced Cost of Ownership
- Single-Hop Capability for Client/Server
- Client/Server Security
- Multisite Data Distribution
- Continuous Querying
- Heterogeneous Data Sharing

High Read-and-Write Throughput

Read-and-write throughput is provided by concurrent main-memory data structures and a highly optimized distribution infrastructure. Applications can make copies of data dynamically in memory through synchronous or asynchronous replication for high read throughput or partition the data across many system members to achieve high read-and-write throughput. Data partitioning doubles the aggregate throughput if the data access is fairly balanced across the entire data set. Linear increase in throughput is limited only by the backbone network capacity.

Low and Predictable Latency

The optimized caching layer minimizes context switches between threads and processes. It manages data in highly concurrent structures to minimize contention points. Communication to peer members is synchronous if the receivers can keep up, which keeps the latency for data distribution to a minimum. Servers manage object graphs in serialized form to reduce the strain on the garbage collector.

Subscription management (interest registration and continuous queries) is partitioned across server data stores, ensuring that a subscription is processed only once for all interested clients. The resulting improvements in CPU use and bandwidth utilization improve throughput and reduce latency for client subscriptions.

High Scalability

Scalability is achieved through dynamic partitioning of data across many members and spreading the data load uniformly across the servers. For “hot” data, you can configure the system to expand dynamically to create more copies of the data. You can also provision application behavior to run in a distributed manner in close proximity to the data it needs.

If you need to support high and unpredictable bursts of concurrent client load, you can increase the number of servers managing the data and distribute the data and behavior across them to provide uniform and predictable response times. Clients are continuously load balanced to the server farm based on continuous feedback from the servers on their load conditions. With data partitioned and replicated across servers, clients can dynamically move to different servers to uniformly load the servers and deliver the best response times.

You can also improve scalability by implementing asynchronous “write behind” of data changes to external data stores, like a database. This avoids a bottleneck by queuing all updates in order and redundantly. You can also conflate updates and propagate them in batch to the database.

Continuous Availability

In addition to guaranteed consistent copies of data in memory, applications can persist data to disk on one or more members synchronously or asynchronously by using a “shared nothing disk architecture.” All asynchronous events (store-forward events) are redundantly managed in at least two members such that if one server fails, the redundant one takes over. All clients connect to logical servers, and the client fails over automatically to alternate servers in a group during failures or when servers become unresponsive.

Reliable Event Notifications

Publish/subscribe systems offer a data-distribution service where new events are published into the system and routed to all interested subscribers in a reliable manner. Traditional messaging platforms focus on message delivery, but often the receiving applications need access to related data before they can process the event. This requires them to access a standard database when the event is delivered, limiting the subscriber by the speed of the database.

Data and events are offered through a single system. Data is managed as objects in one or more distributed data regions, similar to tables in a database. Applications simply insert, update, or delete objects in data regions, and the platform delivers the object changes to the subscribers. The subscriber receiving the event has direct access to the related data in local memory or can fetch the data from one of the other members through a single hop.

Parallelized Application Behavior on Data Stores

You can execute application business logic in parallel on members. The data-aware function-execution service permits execution of arbitrary, data-dependent application functions on the members where the data is partitioned for locality of reference and scale.

By colocating the relevant data and parallelizing the calculation, you increase overall throughput. The calculation latency is inversely proportional to the number of members on which it can be parallelized.

The fundamental premise is to route the function transparently to the application that carries the data subset required by the function and to avoid moving data around on the network. Application function can be executed on only one member, in parallel on a subset of members, or in parallel across all members. This programming model is similar to the popular Map-Reduce model from Google. Data-aware function routing is most appropriate for applications that require iteration over multiple data items (such as a query or custom aggregation function).

Shared-Nothing Disk Persistence

Each cluster member manages data on disk files independent of other members. Failures in disks or cache failures in one member do not affect the ability of another cache instance to operate safely on its disk files. This “shared nothing” persistence architecture allows applications to be configured such that different classes of data are persisted on different members across the system, dramatically increasing the overall throughput of the application even when disk persistence is configured for application objects.

Unlike a traditional database system, separate files are not used to manage data and transaction logs. All data updates are appended to files that are similar to transactional logs of traditional databases. You can avoid disk-seek times if the disk is not concurrently used by other processes, and the only cost incurred is the rotational latency.

Reduced Cost of Ownership

You can configure caching in tiers. The client application process can host a cache locally (in memory and overflow to disk) and delegate to a cache server farm on misses. Even a 30 percent hit ratio on the local cache translates to significant savings in costs. The total cost associated with every single transaction comes from the CPU cycles spent, the network cost, the access to the database, and intangible costs associated with database maintenance. By managing the data as application objects, you avoid the additional cost (CPU cycles) associated with mapping SQL rows to objects.

Single-Hop Capability for Client/Server

Clients can send individual data requests directly to the server holding the data key, avoiding multiple hops to locate data that is partitioned. Metadata in the client identifies the correct server. This feature improves performance and client access to partitioned regions in the server tier.

Client/Server Security

There may be multiple, distinct users in client applications. This feature accommodates installations in which clients are embedded in application servers and each application server supports data requests from many users. Each user may be authorized to access a small subset of data on the servers, as in a customer application where each customer can access only their own orders and shipments. Each user in the client connects to the server with its own set of credentials and has its own access authorization to the server cache.

Multisite Data Distribution

Scalability problems can result from data sites being spread out geographically across a wide-area network (WAN). Models address these topologies, ranging from a single peer-to-peer cluster to reliable communications between data centers across the WAN. This model allows clusters to scale out in an unbounded and loosely coupled fashion without loss of performance, reliability or data consistency.

At the core of this architecture is the gateway sender configuration used for distributing region events to a remote site. You can deploy gateway sender instances in parallel, which enables an increase in throughput for distributing region events across the WAN. You can also configure gateway sender queues for persistence and high availability to avoid data loss in the case of a member failure.

Continuous Querying

In messaging systems like Java Message Service, clients subscribe to topics and queues. Any message delivered to a topic is sent to the subscriber. VMware GemFire allows continuous querying by having applications express complex interest using Object Query Language.

Heterogeneous Data Sharing

C#, C++ and Java applications can share application business objects without going through a transformation layer such as SOAP or XML. The server side behavior, though implemented in Java, provides a unique native cache for C++ and .NET applications. Application objects can be managed in the C++ process heap and distributed to other processes using a common "on-the-wire" representation for objects. A C++ serialized object can be directly deserialized as an equivalent Java or C# object. A change to a business object in one language can trigger reliable notifications in applications written in the other supported languages.

Installing VMware GemFire

This section describes how to install VMware GemFire.

When you install a new version of VMware GemFire software on an existing VMware GemFire system, keep the previous version available until the system upgrade has been completed and verified.

- **Install VMware GemFire from a Compressed TAR File**

  Use the compressed TAR file distribution to install and configure VMware GemFire on every physical and virtual machine where you will run VMware GemFire.

- **Obtaining VMware GemFire from a Maven Repository**

  You can use Maven to add VMware GemFire to your Java project build.
- **Setting Up the CLASSPATH**

  This topic describes how VMware GemFire processes set their CLASSPATH.
- **Uninstalling VMware GemFire**

  This section describes how to remove VMware GemFire.

Obtaining and Installing Security Updates

New versions of VMware GemFire often include important security fixes, so Pivotal recommends you keep up to date with the latest releases.

For details about any security fixes in a particular release, see the Pivotal security page.

Windows/Unix/Linux—Install VMware Tanzu GemFire from a Compressed TAR File

Use the compressed TAR file distribution to install and configure VMware Tanzu GemFire on every physical and virtual machine where you will run VMware Tanzu GemFire.

Prerequisites

Before you install GemFire, you must complete the following prerequisites:

- Confirm that your system meets the hardware and software requirements described in Supported Configurations and System Requirements.
- From the VMware Tanzu GemFire product page on the VMware Tanzu Network, select the version of GemFire to download, then click the box marked **VMware Tanzu GemFire** to download the compressed TAR distribution of GemFire.
- Know how to configure environment variables for your system. If you have not done so already, set the JAVA_HOME environment variable to point to a Java runtime installation supported by GemFire. (You should find a `bin` directory under JAVA_HOME.)

Procedure

Use the following procedure to install VMware Tanzu GemFire:

1. Navigate to the directory where you downloaded the GemFire software, and expand the compressed TAR file after creating the *path_to_product* directory.

   ```
   $ tar -xzvf pivotal-gemfire-N.N.N.tgz -C path_to_product
   ```

   path_to_product corresponds to the location where you want to install GemFire, and *N.N.N* is the version number.
2. Configure the JAVA_HOME environment variable.

If you will be using the `gfsh` command-line utility or managing servers and locators with the `ServerLauncher` and `LocatorLauncher` APIs, then you must set JAVA_HOME to a JDK installation. For example: - **UNIX and Linux (Bourne and Korn shells - sh, ksh, bash)**

~~~
``` pre
JAVA_HOME=/usr/java/jdk1.8.0_272
export JAVA_HOME
```
~~~

- **Windows**

  ```
  set JAVA_HOME="C:\Program Files\Java\jdk1.8.0_272"
  ```

3. This step only applies to environments where you are running GemFire processes or GemFire client applications outside of gfsh. The gfsh script sets these environment variables for you. If you are running GemFire processes or applications outside of gfsh, then configure the following environment variables for GemFire.

   - Set the GEMFIRE environment variable to point to your GemFire installation top-level directory. (You should see `bin`, `lib`, `dtd`, and other directories under GEMFIRE.) The following variables definitions are examples; your installation path will vary, depending on where you install GemFire and the version (*N.N.N*) you are installing.

     - **UNIX and Linux (Bourne and Korn shells - sh, ksh, bash)**

       ```
       GEMFIRE=/opt/pivotal/pivotal-gemfire-N.N.N
       export GEMFIRE
       ```

     - **Windows**

       ```
       set GEMFIRE=C:\pivotal\gemfire\pivotal-gemfire-N.N.N
       ```

   - Configure your GF_JAVA environment variables as shown in these examples. GF_JAVA must point to the `java` executable file under your JAVA_HOME. (If you have not done so already, you should also set your JAVA_HOME variable to a supported Java installation.)

     - **UNIX and Linux (Bourne and Korn shells - sh, ksh, bash)**

       ```
       GF_JAVA=$JAVA_HOME/bin/java
       export GF_JAVA
       ```

     - **Windows**

       ```
       set GF_JAVA=%JAVA_HOME%\bin\java.exe
       ```

4. Add GemFire scripts to your the PATH environment variable. For example:

   - **UNIX and Linux (Bourne and Korn shells - sh, ksh, bash)**

     ```
     PATH=$PATH:$JAVA_HOME/bin:/opt/pivotal/pivotal-gemfire-N.N.N/bin
     export PATH
     ```

- **Windows**

  ```
  set PATH=%PATH%;%JAVA_HOME%\bin;%GEMFIRE%\bin
  ```

5. Type `gfsh version` at the command line and verify that the output lists the version of VMware Tanzu GemFire that you wish to install. For example:

   ```
   $ gfsh version
   v9.0.0
   ```

   If you want more detailed version information such as the date of the build, build number and JDK version being used, type `gfsh version --full`.

6. Repeat this procedure for every virtual or physical machine on which you will run VMware Tanzu GemFire.

Obtaining Pivotal GemFire from a Maven Repository

You can use Maven to add Pivotal GemFire to your Java project build.

1. Access to the Pivotal Commercial Maven Repository requires a one-time registration step to obtain an account. The URL for both registration and subsequent logins after registration is https://commercial-repo.pivotal.io/login/auth. Click on the Create Account link to register. You will receive a confirmation email; follow the directions in this email to activate your account.

2. After account activation, log in at https://commercial-repo.pivotal.io/login/auth to access the configuration information in the Pivotal GemFire Release Repository.

3. To add GemFire to your Java project, you need to modify your project's `pom.xml` file. Add the following repository definition to your `pom.xml` file:

   ```
      <repository>
          <id>gemfire-release-repo</id>
          <name>Pivotal GemFire Release Repository</name>
          <url>https://commercial-repo.pivotal.io/data3/gemfire-release-repo/gemfi
   re</url>
      </repository>
   ```

4. Add the following dependencies to your `pom.xml` file:

   ```
   <dependencies>
      <dependency>
         <groupId>io.pivotal.gemfire</groupId>
         <artifactId>geode-core</artifactId>
         <version>VERSION-ID</version>
      </dependency>
      <dependency>
         <groupId>io.pivotal.gemfire</groupId>
         <artifactId>geode-wan</artifactId>
         <version>VERSION-ID</version>
      </dependency>
      <dependency>
         <groupId>io.pivotal.gemfire</groupId>
         <artifactId>geode-cq</artifactId>
   ```

```
        <version>VERSION-ID</version>
    </dependency>
</dependencies>
```

where *VERSION-ID* is the version identifier of GemFire that you wish to install. For example, **9.0.0**.

5. To access these artifacts, you must add an entry to your `.m2/settings.xml` file:

```
<settings>
    <servers>
        <server>
            <id>gemfire-release-repo</id>
            <username>MY-USERNAME@example.com</username>
            <password>MY-DECRYPTED-PASSWORD</password>
        </server>
    </servers>
</settings>
```

where you provide the values for `MY-USERNAME@example.com` and `MY-DECRYPTED-PASSWORD`. As of Maven version 2.1.0, encrypted passwords are supported in this `settings.xml` file. See https://maven.apache.org/guides/mini/guide-encryption.html for details on the encryption.

Setting Up the CLASSPATH

This topic describes how VMware GemFire processes set their CLASSPATH.

To simplify CLASSPATH environment settings, VMware GemFire has organized all application libraries required by VMware GemFire processes into `*-dependencies.jar` files. All dependency JAR files are located in the `path_to_product/lib` directory.

When starting a server or locator process using `gfsh`, application JAR files are automatically loaded into the process's CLASSPATH from two directories:

- `path_to_product/lib/`
- `path_to_product/extensions/`

Note: To embed VMware GemFire in your application, add `path_to_product/lib/geode-dependencies.jar` to your CLASSPATH.

The following table lists the dependency JAR files associated with various VMware GemFire processes:

VMware GemFire Process	Associated JAR Files
gfsh	gfsh-dependencies.jar
server and locator	geode-dependencies.jar Note: Use this library for all standalone or embedded VMware GemFire processes (including Java clients) that host cache data.

Modifying the CLASSPATH in gfsh-Managed Processes

There are two options for updating the CLASSPATH of VMware GemFire server and locator processes that are started on the gfsh command line.

Option 1: Specify the `--classpath` parameter upon process startup. For example, to modify the CLASSPATH of a locator:

```
gfsh> start locator --name=locator1 --classpath=/path/to/applications/classes.jar
```

And to modify the CLASSPATH of a server:

```
gfsh> start server --name=server1 --classpath=/path/to/applications/classes.jar
```

Application classes supplied as arguments to the `--classpath` option are *prepended* to the server or locator's CLASSPATH, beginning in second position. The first entry in the CLASSPATH is reserved for the core VMware GemFire jar file, for security reasons.

Option 2: Define the CLASSPATH environment variable in your OS environment. Then, specify the `--include-system-classpath` parameter upon process startup. For example:

```
gfsh> start locator --name=locator1 --include-system-classpath=true
```

The same can also be done for server processes:

```
gfsh> start server --name=server1 --include-system-classpath=true
```

This option *appends* the contents of the system CLASSPATH environment variable to the locator or server's CLASSPATH upon startup. Specifying this option without a value sets it to true.

Setting the CLASSPATH for Applications and Standalone Java Processes

If you are starting a VMware GemFire process programmatically (standalone or embedded), we recommend that you specify the CLASSPATH upon program execution using the `java -classpath` or `java -cp` command-line option. This method is preferred to setting the CLASSPATH as an environment variable since it allows you to set the value individually for each application without affecting other applications and without other applications modifying its value.

For example, to start up a VMware GemFire locator process using the LocatorLauncher API, you can execute the following on the command line:

```
prompt# java -cp "path_to_product/lib/geode-dependencies.jar"
org.apache.geode.distributed.LocatorLauncher start locator1
<locator-launcher-options>
```

To start up a VMware GemFire server process using the ServerLauncher API:

```
prompt# java -cp "path_to_product/lib/geode-dependencies.jar:/path/to/your/application
s/classes.jar"
org.apache.geode.distributed.ServerLauncher start server1
<server-launcher-options>
```

Note that in addition to the `*-dependencies.jar` file associated with the process, you must also specify any custom application JARs that you wish to access in your VMware GemFire process. For example, if you are planning on using a customized compressor on your regions, you should specify the application JAR that contains the compressor application you wish to use.

To start up an application with an embedded cache:

```
java -cp "path_to_product/lib/geode-dependencies.jar:/path/to/your/applications/classe
s.jar"
com.mycompany.package.ApplicationWithEmbeddedCache
```

Note: Another method for updating the CLASSPATH of a server process with your own applications is to use the `gfsh deploy` command. Deploying application JAR files will automatically update the CLASSPATH of all members that are targeted for deployment. See Deploying Application JARs to VMware GemFire Members for more details.

For systems running an embedded HTTP or HTTPS service, setting a `GEODE_HOME` environment variable with a path to the VMware GemFire installation directory allows a server launcher to find the WAR file without any changes to the CLASSPATH.

Uninstalling GemFire

This section describes how to remove GemFire from your system.

If you installed Pivotal GemFire from a ZIP file, shut down any running GemFire processes and then simply delete the product tree to uninstall the product. No additional registry entries or system modifications are needed.

If the old version of GemFire was installed with an RPM, uninstall using

```
rpm -e Pivotal_GemFire_XXX
```

where `XXX` is replaced by the GemFire version number and also corresponds to the name of the product installation directory. As an example, the command for removing the GemFire 8.2.5 release would be

```
rpm -e Pivotal_GemFire_825
```

If the old version of GemFire was installed with DEBs, uninstall using

```
dpkg --remove pivotal-gemfire
```

If you installed Pivotal GemFire by using macOS brew packages, issue the following command:

```
brew uninstall gemfire
```

Upgrading VMware GemFire

To upgrade an existing installation to a new version of VMware GemFire, follow these general steps:

1. Back up your current system.

2. Install the new version of the software.
3. Stop your distributed system using the current software.
4. Restart the system using the new software.

In many cases, components running under the current version can be stopped selectively, then restarted under the new version so that the distributed system as a whole remains functional during the upgrade process; this is known as a “rolling upgrade.”

In other cases, the entire system must be stopped in order to accomplish the upgrade, as when upgrading from one major version to another (for example, from GemFire 8.2 to 9.0), which will require some downtime for your system.

See Planning an Upgrade to choose the upgrade scenario that best suits your implementation and to understand the resources you will need to accomplish the upgrade. Then select the appropriate upgrade procedure for more detailed instructions that fit your specific needs.

Upgrade Details

- **Planning an Upgrade**

  This section discusses the upgrade paths for various VMware GemFire versions, and it lists information you need to know before you begin the upgrade process.

- **Rolling Upgrade**

  A rolling upgrade allows you to keep your existing distributed system running while you upgrade your members gradually.

- **Offline Upgrade**

  An offline upgrade can handle the widest variety of software versions and cluster configurations, but requires shutting down the entire system for at least a short time.

- **Upgrading Clients**

- **Upgrade from Version 8.2 to Version 9**

  Details on how to upgrade GemFire from version 8.2.3 or a more recent 8.2 version to version 9.

- **Multi-site Upgrade from Version 8.2 to Version 9**

  A special procedure allows some multi-site systems to upgrade to version 9 on a site-by-site basis, eliminating the need for system-wide down time.

Planning an Upgrade

Before you upgrade your system, back it up. Make backup copies of all existing disk-stores, server-side code, configuration files, and data across the entire cluster. To get a backup of the data that includes the most recent changes may require that traffic across the cluster is stopped before the backup is made. The discussion at Creating Backups for System Recovery and Operational Management explains the process, and the backup disk-store command reference page describes how to use the `gfsh backup disk-store` command to make a backup.

Guidelines for Upgrading

- Schedule your upgrade during a period of low user activity for your system and network.
- **Important:** After all locators have been upgraded, *do not start or restart any processes* that use the older version of the software. The older process will either not be allowed to join the distributed system or, if allowed to join, can potentially cause a deadlock.
- Verify that all members that you wish to upgrade are members of the same distributed system cluster. A list of cluster members will be output with the `gfsh` command:

  ```
  gfsh>list members
  ```

- Locate a copy of your system's startup script, if your site has one (most do). The startup script can be a handy reference for restarting upgraded locators and servers with the same `gfsh` command lines that were used in your current installation.
- Identify how your current cluster configuration was specified. The way in which your cluster configuration was created determines which commands you use to save and restore that cluster configuration during the upgrade procedure. There are two possibilites:
  - With `gfsh` commands, relying on the underlying **cluster configuration service** to record the configuration: see Exporting and Importing Cluster Configurations.
  - With **XML properties** specified through the Java API or configuration files: see Deploying Configuration Files without the Cluster Configuration Service.
- Do not modify region attributes or data, either via `gfsh` or `cache.xml` configuration, during the upgrade process.

Version Compatibilities

Your choice of upgrade procedure depends, in part, on the versions of VMware GemFire involved.

- **Version Compatibility Between Peers and Cache Servers**

  For best reliability and performance, all server components of a VMware GemFire system should run the same version of the software. For the purposes of a rolling upgrade, you can have peers or cache servers running different minor versions of VMware GemFire at the same time, as long as the major version is the same. For example, some components can continue to run under version 9.0 while you are in the process of upgrading to version 9.1.

- **Version Compatibility Between Clients and Servers**

  VMware GemFire clients can run version 8.2.3 or a more recent 8.2 version of VMware GemFire and still connect to VMware GemFire servers running version 9.x. Version 9.x clients, however, cannot connect to servers running older versions of VMware GemFire.

- **Version Compatibility Between Sites in Multi-Site (WAN) Deployments**

  In multi-site (WAN) deployments, one site can be running VMware GemFire 8.2.3 or a more recent VMware GemFire 8.2 version, and another site can be running VMware GemFire 9.x. The sites should still be able to communicate with one another.

Upgrade to the Latest Version 9 from an Earlier Version 9

If possible, follow the Rolling Upgrade procedure. A multi-site installation can also do rolling upgrades within each site. If a rolling upgrade is not possible, follow the Off-Line Upgrade procedure. A rolling upgrade is not possible for a cluster that has partitioned regions without redundancy. Without the redundancy, region entries will be lost when individual servers are taken out of the cluster during a rolling upgrade.

Upgrade to Version 9 from Version 8.2.3 or a More Recent 8.2 Version

To upgrade all servers from version 8.2.3 or a more recent version of 8.2 to this version of Pivotal VMware GemFire 9, follow the Upgrade from Version 8.2 to Version 9 procedure.

Upgrade to Version 9 from Version 8.2.2 or an Earlier Version

All upgrades to this VMware GemFire version 9 from VMware GemFire versions earlier than version 8.2.3 follow a two-step process:

1. Upgrade all servers to the most recent version of VMware GemFire version 8.2 (which must be 8.2.3 or later). If possible, follow the Rolling Upgrade procedure. If a rolling upgrade is not possible, follow the Off-Line Upgrade procedure.

   A rolling upgrade is not possible for a cluster that has partitioned regions without redundancy. Without the redundancy, region entries will be lost when individual servers are taken out of the cluster during a rolling upgrade.

2. Upgrade all servers from the most recent version of 8.2 to this version of VMware GemFire 9. Follow the Upgrade from Version 8.2 to Version 9 procedure.

Upgrade a Multi-Site System to Version 9 from Version 8.2.3 or Later

Multi-site systems that have both persistent disk stores and use bidirectional WAN gateways to replicate data among sites may be able to avoid taking the entire system down to upgrade to version 9. This Multi-site Upgrade from Version 8.2 to Version 9 procedure requires multiple reconfigurations during the upgrade as one site at a time is upgraded.

Java Notes

- To check your current Java version, type `java -version` at a command-line prompt.
- VMware GemFire 9.x requires Java SE 8, version 92 or a more recent version. VMware GemFire 8.0 was the last VMware GemFire release to support Java SE 6, and Java SE 7 is in End-of-Life status.
- The VMware GemFire product download does not include Java. You must download and install a supported JRE or JDK on each system running VMware GemFire. To obtain best performance with commands such as `gfsh status` and `gfsh stop`, install a full JDK (not just a JRE).

Rolling Upgrade

A rolling upgrade eliminates system downtime by keeping your existing distributed system running while you upgrade one member at a time. Each upgraded member can communicate with other members that are still running the earlier version of GemFire, so servers can respond to client requests even as the upgrade is underway. Interdependent data members can be stopped and started without mutually blocking, a problem that can occur when multiple data members are stopped at the same time.

Rolling Upgrade Limitations and Requirements

Versions

Rolling upgrade requires that the older and newer versions of VMware GemFire are mutually compatible, which usually means that they share the same major version number. Therefore, you can perform a rolling upgrade to upgrade from:

- Earlier versions of 9.x up to the most recent version of 9.10.
- Earlier versions of 8.x up to the most recent version of 8.2.

See Version Compatibilities for more details on how different versions of GemFire can interoperate.

Components

Rolling upgrades apply to the peer members or cache servers within a distributed system. Under some circumstances, rolling upgrades can also be applied within individual sites of multi-site (WAN) deployments.

Redundancy

All partitioned regions in your system must have full redundancy. Check the redundancy state of all your regions *before* you begin the rolling upgrade and *before* stopping any members. See Checking Redundancy in Partitioned Regions for details.

If a rolling update is not possible for your system, follow the Off-Line Upgrade procedure.

Rolling Upgrade Guidelines

Do not create or destroy regions

When you perform a rolling upgrade, your online cluster will have a mix of members running different versions of GemFire. During this time period, do not execute region operations such as region creation or region destruction.

Region rebalancing affects the restart process

If you have `startup-recovery-delay` deactivated (set to -1) for your partitioned region, you must perform a rebalance on your region after you restart each member. If rebalance occurs automatically, as it will if `startup-recovery-delay` is enabled (set to a value other than -1), make sure that the rebalance completes before you stop the next server. If you have `startup-recovery-delay` enabled and set to a high number, you may need to wait extra time until the region has recovered redundancy, because rebalance must complete before new servers are restarted. The partitioned region attribute `startup-recovery-delay` is described in Configure Member Join Redundancy Recovery for a Partitioned Region.

Checking component versions while upgrading

During a rolling upgrade, you can check the current GemFire version of all members in the cluster by looking at the server or locator logs.

When an upgraded member reconnects to the distributed system, it logs all the members it can see as well as the GemFire version of those members. For example, an upgraded locator will now detect GemFire members running the older version of GemFire (in this case, the version being upgraded– GFE 9.0.0) :

```
[info 2013/06/03 10:03:29.206 PDT frodo <vm_1_thr_1_frodo> tid=0x1a]  DistributionMana
ger frodo(locator1:21869:locator)<v16>:28242 started on frodo[15001]. There
    were 2 other DMs. others: [frodo(server2:21617)<v4>:14973( version:GFE 9.0.0 ), fr
odo(server1:21069)<v1>:60929( version:GFE 9.0.0 )] (locator)
```

After some members have been upgraded, non-upgraded members will log the following message when they receive a new membership view:

```
Membership: received new view [frodo(locator1:20786)<v0>:32240|4]
      [frodo(locator1:20786)<v0>:32240/51878, frodo(server1:21069)<v1>:60929/46949,
      frodo(server2:21617)<v4>( version:UNKNOWN[ordinal=23] ):14973/33919]
```

Non-upgraded members identify members that have been upgraded to the next version with `version: UNKNOWN`.

Cluster configuration affects save and restore

The way in which your cluster configuration was created determines which commands you use to save and restore that cluster configuration during the upgrade procedure.

- If your system was configured with `gfsh` commands, relying on the underlying **cluster configuration service**, the configuration can be saved in one central location, then applied to all newly-upgraded members. See Exporting and Importing Cluster Configurations.
- If your system was configured with **XML properties** specified through the Java API or configuration files, you must save the configuration for each member before you bring it down, then re-import it for that member's upgraded counterpart. See Deploying Configuration Files without the Cluster Configuration Service.

Rolling Upgrade Procedure

Begin by installing the new version of the software alongside the older version of the software on all hosts. You will need both versions of the software during the upgrade procedure. See Installing Pivotal GemFire.

Upgrade locators first, then data members, then clients.

Upgrade Locators

1. On the machine hosting the first locator you wish to upgrade, open a terminal console.
2. Start a `gfsh` prompt, using the version from your current GemFire installation, and connect to the currently running locator. For example:

```
gfsh>connect --locator=locator_hostname_or_ip_address[port]
```

3. Use `gfsh` commands to characterize your current installation so you can compare your post-upgrade system to the current one. For example, use the `list members` command to view locators and data members:

```
  Name   | Id
-------- | ------------------------------------------------
locator1 | 172.16.71.1(locator1:26510:locator)<ec><v0>:1024
locator2 | 172.16.71.1(locator2:26511:locator)<ec><v1>:1025
server1  | 172.16.71.1(server1:26514)<v2>:1026
server2  | 172.16.71.1(server2:26518)<v3>:1027
```

4. Save your cluster configuration.
   - If you are using the cluster configuration service, use the gfsh `export cluster-configuration` command. You only need to do this once, as the newly-upgraded locator will propagate the configuration to newly-upgraded members as they come online.
   - For an XML configuration, save `cache.xml`, `gemfire.properties`, and any other relevant configuration files to a well-known location. You must repeat this step for each member you upgrade.
5. Stop the locator. For example:

```
gfsh>stop locator --name=locator1
Stopping Locator running in /Users/username/sandbox/locator on 172.16.71.1[1033
4] as locator...
Process ID: 96686
Log File: /Users/username/sandbox/locator/locator.log
....
No longer connected to 172.16.71.1[1099].
```

6. Start `gfsh` from the new GemFire installation. Verify that you are running the newer version with

```
gfsh>version
```

7. Start a locator and import the saved configuration. If you are using the cluster configuration service, use the same name and directory as the older version you stopped, and the new locator will access the old locator's cluster configuration without having to import it in a separate step:

```
gfsh>start locator --name=locator1 --enable-cluster-configuration=true --dir=/d
ata/locator1
```

   Otherwise, use the gfsh `import cluster-configuration` command or explicitly import `.xml` and `.properties` files, as appropriate.

8. The new locator should reconnect to the same members as the older locator. Use `list members` to verify:

```
gfsh>list members
  Name    | Id
-------- | -----------------------------------------------------
locator1 | 172.16.71.1(locator1:26752:locator)<ec><v17>:1024(version:UNKNOWN[or
dinal=65])
locator2 | 172.16.71.1(locator2:26511:locator)<ec><v1>:1025
server1  | 172.16.71.1(server1:26514)<v2>:1026
server2  | 172.16.71.1(server2:26518)<v3>:1027
```

9. Upgrade the remaining locators by stopping and restarting them. When you have completed that step, the system gives a more coherent view of version numbers:

```
gfsh>list members
  Name    | Id
-------- | -----------------------------------------------------
locator1 | 172.16.71.1(locator1:26752:locator)<ec><v17>:1024
locator2 | 172.16.71.1(locator2:26808:locator)<ec><v30>:1025
server1  | 172.16.71.1(server1:26514)<v2>:1026(version:GFE 9.0)
server2  | 172.16.71.1(server2:26518)<v3>:1027(version:GFE 9.0)
```

The server entries show that the servers are running an older version of gemfire, in this case `(version:GFE 9.0)`.

Upgrade Servers

After you have upgraded all of the system's locators, upgrade the servers.

1. Upgrade each server, one at a time, by stopping it and restarting it. Restart the server with the same command-line options with which it was originally started in the previous installation. For example:

```
gfsh>stop server --name=server1
Stopping Cache Server running in /Users/share/server1 on 172.16.71.1[52139] as
server1...

gfsh>start server --name=server1 --use-cluster-configuration=true --server-port
=0 --dir=/data/server1
Starting a Geode Server in /Users/share/server1...
```

Use the `list members` command to verify that the server is now running the new version of GemFire:

```
gfsh>list members
  Name    | Id
-------- | -----------------------------------------------------
locator1 | 172.16.71.1(locator1:26752:locator)<ec><v17>:1024
locator2 | 172.16.71.1(locator2:26808:locator)<ec><v30>:1025
server1  | 172.16.71.1(server1:26835)<v32>:1026
server2  | 172.16.71.1(server2:26518)<v3>:1027(version:GFE 9.0)
```

2. Restore data to the data member. If automatic rebalancing is enabled (partitioned region attribute `startup-recovery-delay` is set to a value other than -1), data restoration will start automatically. If automatic rebalancing is deactivated (partitioned region attribute `startup-recovery-delay=-1`), you must initiate data restoration by issuing the gfsh `rebalance` command.

Wait until the newly-started server has been restored before upgrading the next server. You can repeat various gfsh `show metrics` command with the `--member` option or the `--region` option to verify that the data member is hosting data and that the amount of data it is hosting has stabilized.

3. Shut down,restart, and rebalance servers until all data members are running the new version of GemFire.

Upgrade Clients

Upgrade VMware GemFire clients, following the guidelines described in Upgrading Clients.

Offline Upgrade

Use the offline upgrade procedure when you cannot, or choose not to, perform a rolling upgrade. For example, a rolling upgrade is not possible for a cluster that has partitioned regions without redundancy. (Without the redundancy, region entries would be lost when individual servers were taken out of the cluster during a rolling upgrade.)

Offline Upgrade Guidelines

Versions

For best reliability and performance, all server components of a VMware GemFire system should run the same version of the software. See Version Compatibilities for more details on how different versions of GemFire can interoperate.

Data member interdependencies

When you restart your upgraded servers, interdependent data members may hang on startup waiting for each other. In this case, start the servers in separate command shells so they can start simultaneously and communicate with one another to resolve dependencies.

Offline Upgrade Procedure

1. Stop any connected clients.
2. On a machine hosting a locator, open a terminal console.
3. Start a `gfsh` prompt, using the version from your current GemFire installation, and connect to a currently running locator. For example:

```
gfsh>connect --locator=locator_hostname_or_ip_address[port]
```

4. Use `gfsh` commands to characterize your current installation so you can compare your post-upgrade system to the current one. For example, use the `list members` command to view locators and data members:

```
  Name    | Id
-------- | -------------------------------------------------
locator1 | 172.16.71.1(locator1:26510:locator)<ec><v0>:1024
locator2 | 172.16.71.1(locator2:26511:locator)<ec><v1>:1025
```

```
server1   | 172.16.71.1(server1:26514)<v2>:1026
server2   | 172.16.71.1(server2:26518)<v3>:1027
```

5. Save your cluster configuration.
   - If you are using the cluster configuration service, use the gfsh `export cluster-configuration` command. You only need to do this once, as the newly-upgraded locator will propagate the configuration to newly-upgraded members as they come online.
   - For an XML configuration, save `cache.xml`, `gemfire.properties`, and any other relevant configuration files to a well-known location. You must repeat this step for each member you upgrade.
6. Shut down the entire cluster (by pressing Y at the prompt, this will lose no persisted data):

   ```
   gfsh>shutdown --include-locators=true
   As a lot of data in memory will be lost, including possibly events in queues, d
   o you really want to shutdown the entire distributed system? (Y/n): y
   Shutdown is triggered

   gfsh>
   No longer connected to 172.16.71.1[1099].
   gfsh>quit
   ```

   Since GemFire is a Java process, to check before continuing that all GemFire members successfully stopped, it is useful to use the JDK-included `jps` command to check running java processes:

   ```
   % jps
   29664 Jps
   ```

7. On each machine in the cluster, install the new version of the software (alongside the older version of the software). See Installing Pivotal GemFire.
8. Redeploy your environment's configuration files to the new version installation. If you are using the cluster configuration service, one copy of the exported `.zip` configuration file is sufficient, as the first upgraded locator will propagate it to the other members. For XML configurations, you should have a copy of the saved configuration files for each data member.
9. On each machine in the cluster, install any updated server code. Point all client applications to the new installation of GemFire.
10. Run the new version of `gfsh`.
11. Start a locator and import the saved configuration. If you are using the cluster configuration service, use the same name and directory as the older version you stopped, and the new locator will access the old locator's cluster configuration without having to import it in a separate step:

    ```
    gfsh>start locator --name=locator1 --enable-cluster-configuration=true --dir=/d
    ata/locator1
    ```

Otherwise, use the gfsh `import cluster-configuration` command or explicitly import `.xml` and `.properties` files, as appropriate.

12. Restart all system data members using the new version of gfsh. Use the same names, directories, and other properties that were used when starting the system under the previous version of the software. (Here is where a system startup script comes in handy as a reference.) Interdependent data members may hang on startup waiting for each other. In this case, start servers in separate shells so they can communicate with one another to resolve dependencies.
13. Upgrade VMware GemFire clients, following the guidelines described in Upgrading Clients.

Upgrading Clients

When you upgrade your GemFire server software, you will likely need to update your client applications in order to maintain compatibility with the upgraded servers. To support real-world implementations, servers can usually interoperate with a few different versions of the client software. In general, you will have best performance and reliability if:

- All clients run the same version of the client software.
- Clients and servers both run the latest versions of their respective software.

GemFire server and native client software releases follow similar numbering schemes, but they are not released in lockstep. For more detailed information regarding version compatibility, see the Native Client for VMware GemFire product documentation.

Changes you may need to make when you update, recompile, and link your client code include:

- Removing or replacing obsolete identifiers
- Renaming packages
- Reinstating secure client/server messaging

Remove or Replace Obsolete Identifiers

Review the Release Notes for a list of classes, methods, and other identifiers that are no longer present in the current release. Update client code so it no longer uses any of these removed identifiers.

Rename Packages

Pivotal GemFire 9 uses the same packages as open-source Apache Geode. Beginning with Gemfire version 9.0, `com.gemstone.gemfire` package names are now `org.apache.geode`. The server class path uses `geode-dependencies.jar` in place of the no-longer-used `gemfire.jar` and `server-dependencies.jar`. If you have written code that explicitly imports `gemfire` packages, you must change those references to use the `geode` names and recompile.

For C++ clients, update the namespace to use `apache::geode::client` in place of `gemfire`.

For .NET clients, use `Apache.Geode.Client` in place of `Gemstone.GemFire.Cache.Generic`.

The Pivotal GemFire 9.x release is backwards compatible with Pivotal GemFire 8.2.3 and more recent 8.2 clients, so a version 9 server will understand calls from an 8.2.3 client that uses the old `com.gemstone.gemfire` package names. There is one exception: Pivotal GemFire 8.x clients that run functions on servers using the `Execution` interface's `execute(Function function)` signature cannot work with GemFire 9.x servers. Clients that run server-side functions with this signature must be reimplemented as 9.x clients that use the 9.x package names.

Reinstate Secure Client/Server Messaging After Upgrading

When servers are upgraded to GemFire v9.1.1 or later from an earlier release, their authentication of client messages is deactivated to facilitate rolling upgrades. If your clients are configured to send authenticated messages, the servers will honor those messages but will not enforce authentication until you reinstate authentication on each of the upgraded servers. To reinstate secure client/server messaging, restart each server with the `geode.disallow-internal-messages-without-credentials` system property set to `true`. For example:

```
gfsh>start server --name=server_name --dir=server_config_dir \
--J=-Dgeode.disallow-internal-messages-without-credentials=true
```

Upgrade from Version 8.2 to Version 9

This is the upgrade procedure to a GemFire 9 version from Pivotal GemFire 8.2.3 or a more recent release of Pivotal GemFire 8.2.

General Upgrade Steps

These steps identify the ordering of an upgrade. A more detailed procedure is given below.

1. Implement package renaming for server-side callbacks in a test environment. The server class path uses `geode-dependencies.jar` in place of the no longer used `gemfire.jar` and `server-dependencies.jar`. Change your server-side callbacks and functions to use `org.apache.geode` packages instead of `com.gemstone.gemfire` packages. Also implement package renaming for any client applications that are to be upgraded to a Pivotal GemFire 9 version.
2. Make backups, so you can restore your GemFire version 8 cluster if the upgrade does not complete to your satisfaction.
    - Back up all existing disk-stores and data.
    - Back up all configuration files including `gemfire.properties` and `.xml` configurations.
3. Check system requirements and make changes where applicable.
4. Shut down the cluster.
5. Upgrade the Java JDK to v1.8.0_92 or later, if necessary.
6. Install the new version of GemFire, and update environment variables to point to the new installation.
7. Install revised server-side callbacks and any upgraded clients.
8. Start the cluster using the new GemFire version 9 installation.
9. Test the new installation to assure that the cluster has been properly redeployed.

10. Redeploy client applications.
11. Test client applications for proper functionality.
12. Remove the old distribution to prevent inadvertent version conflicts.

Java Notes

- The Pivotal GemFire product download does not include Java. You must download and install a supported JRE or JDK on each system running GemFire. GemFire recommends the installation of a full JDK (and not just a JRE) to obtain better performance with `gfsh status` and `gfsh stop` commands.
- To check your current Java version, type `java -version` at a command-line prompt.
- Pivotal GemFire 9.x requires Java SE 8, version 92 or a more recent version. Pivotal GemFire 8.0 was the last GemFire release to support Java SE 6, and Java SE 7 is in End-of-Life status.

RHEL/Centos: with previous installation via RPM

RPMs existed for GemFire 8.x, but they are not available for GemFire 9.x. To install on a RHEL/Centos system, follow these upgrade procedure instructions, with a modified final step of uninstalling the GemFire 8.x version using `rpm -e`.

Ubuntu: with previous installation via Debian packaging

DEBs existed for GemFire 8.x, but they are not available for GemFire 9.x. To install on an Ubuntu system, follow the upgrade procedure instructions, with a modified final step of uninstalling the GemFire 8.x version using `dpkg --remove` or `dpkg --purge`.

Package Renaming

Pivotal GemFire 9.x uses the same packages as open-source Apache Geode. Pivotal GemFire 8.x did not use the Apache Geode package naming. All `com.gemstone.gemfire` package names are now `org.apache.geode`. If you have written code that explicitly imports `gemfire` packages, you must change those references to use the `geode` names. This applies to all server side code. Search and replace `com.gemstone.gemfire` with `org.apache.geode`.

The Pivotal GemFire 9.x release is backwards compatible with Pivotal GemFire 8.2.3 and more recent 8.2 clients, so a version 9 server will understand calls from an 8.2.3 client that uses the old `com.gemstone.gemfire` package names. Pivotal GemFire 9.x peers and servers are *not* compatible with Pivotal GemFire 8.2.3 peers or servers.

Pivotal GemFire 8.2.3 and more recent version 8.2 clients that run functions on servers using the `Execution` interface's `execute(Function function)` signature can not work with 9.x servers. The clients that run functions with this signature must be reimplemented as 9.x clients which use the 9.x package names.

Any backups or exports using data from a version prior to 8.2.3 cannot be used to restore that backup, or import that data, into any GemFire version 9.x or later cluster.

The Upgrade Procedure, Step by Step

Follow these steps to upgrade to Pivotal GemFire 9.x on Linux, Unix, or Windows.

1. Implement package renaming for server-side callbacks in a test environment. The server class path uses `geode-dependencies.jar` in place of the no longer used `gemfire.jar` and `server-dependencies.jar`. Change your server-side callbacks and functions to use `org.apache.geode` packages instead of `com.gemstone.gemfire` packages. Also implement package renaming for any client applications that are to be upgraded to a Pivotal GemFire 9 version.
2. Make backup copies of all existing disk-stores, server-side code, configuration files, and data across the entire cluster. To get a backup of the data that includes the most recent changes may require that traffic across the cluster is stopped before the backup is made.
3. Shut down the cluster running with the prior version. Open a `gfsh` prompt:

   ```
   % gfsh
   ```

   Connect to the locator:

   ```
   gfsh>connect --locator=localhost[10334]
   Connecting to Locator at [host=localhost, port=10334] ..
   Connecting to Manager at [host=192.0.2.0, port=1099] ..
   Successfully connected to: [host=192.0.2.0, port=1099]

   gfsh>list members
    Name    | Id
   ------- | --------------------------------------------
   server2 | 192.0.2.0(server2:29368)<v2>:35840
   locator | 192.0.2.0(locator:29181:locator):36278
   server1 | 192.0.2.0(server1:29285)<v1>:40574
   ```

   Shut down the entire cluster (by pressing Y at the prompt, this will lose no persisted data):

   ```
   gfsh>shutdown --include-locators=true
   ```

As a lot of data in memory will be lost, including possibly events in queues, do you really want to shutdown the entire distributed system? (Y/n): Y succeeded in shutting down ```

```
Since GemFire is a Java process,
```

to check before continuing that all GemFire members successfully stopped, it is useful to use the JDK-included `jps` command to check running java processes:

~~~
``` pre
% jps
29664 Jps
```
~~~

1. On each machine in the cluster, upgrade Java, if needed.
2. On the machine hosting the locator you wish to upgrade, install the new version of the software (alongside the older version of the software). See Installing Pivotal GemFire.

3. On each machine in the cluster, install the new version of the software (alongside the older version of the software). See Installing Pivotal GemFire.

   To check that the system finds the new installation, open a `gfsh` prompt and check the GemFire version:

```
% gfsh --version
v9.0.0
```

4. On each machine in the cluster, redeploy your environment's configuration files to the new version installation.
5. Restart all system members according to your usual procedures.
6. Upgrade VMware GemFire clients. See Upgrading Clients for details.
7. Once all systems are functioning normally and all tests are successful, remove the old version of GemFire from each machine in the cluster, to reduce possibility of version complications in the future. See Uninstalling GemFire for instructions.

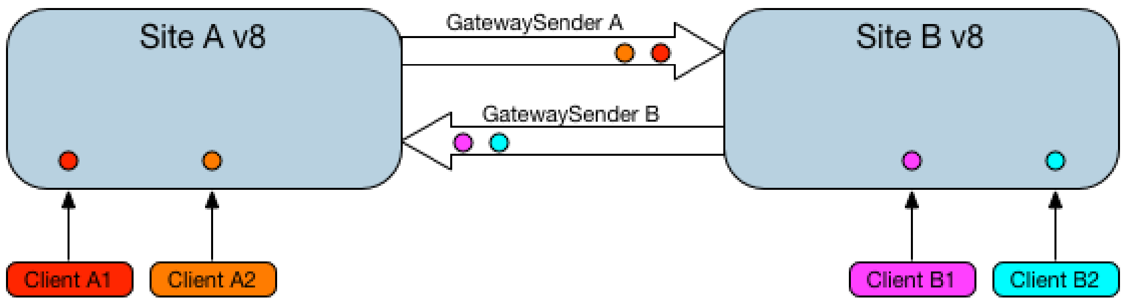
Multi-site Upgrade from Version 8.2 to Version 9

Systems that have both persistent disk stores and a bidirectional WAN gateway to replicate data among sites can implement an upgrade that does not require system down time. It requires multiple reconfigurations during the process, so that each site can be individually upgraded while the rest of the system continues.

This procedure will not work reliably for you if any of your WAN-connected systems includes clients that use continuous queries or subscriptions. In those cases, upgrade each system separately using the procedure described in Upgrade from Version 8.2 to Version 9.

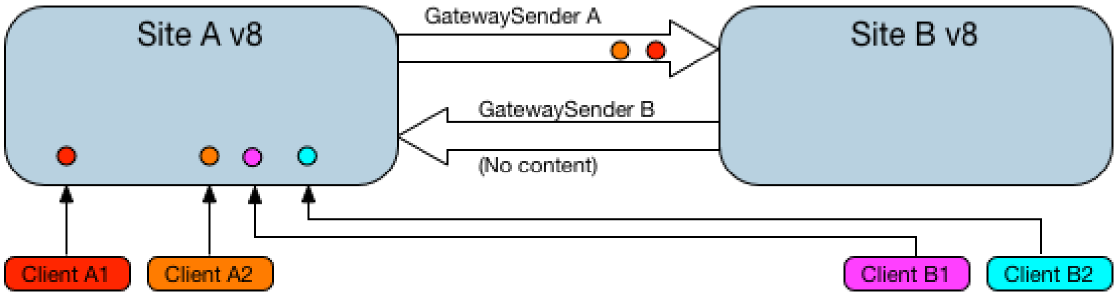
This procedure assumes that each site has already been upgraded to Pivotal GemFire version 8.2.3 or a more recent 8.2 version.

This diagram shows the initial state of the system.

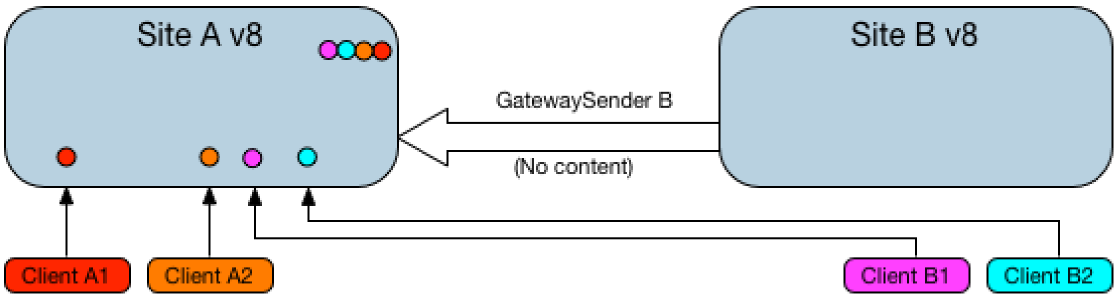


The procedure for upgrading two sites, Site A and Site B:

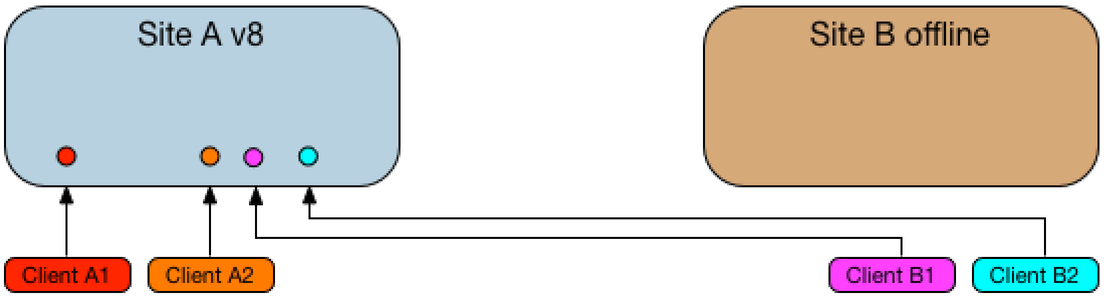
1. Reconfigure to redirect all traffic away from Site B (which will be the first site that gets upgraded).



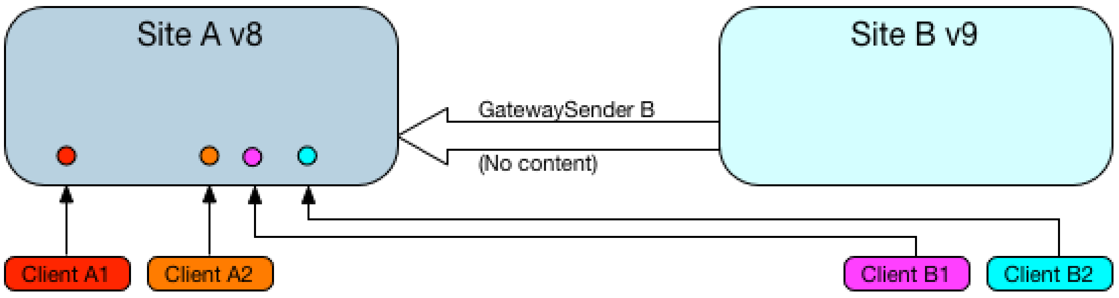
2. Pause the gateway senders on Site A.



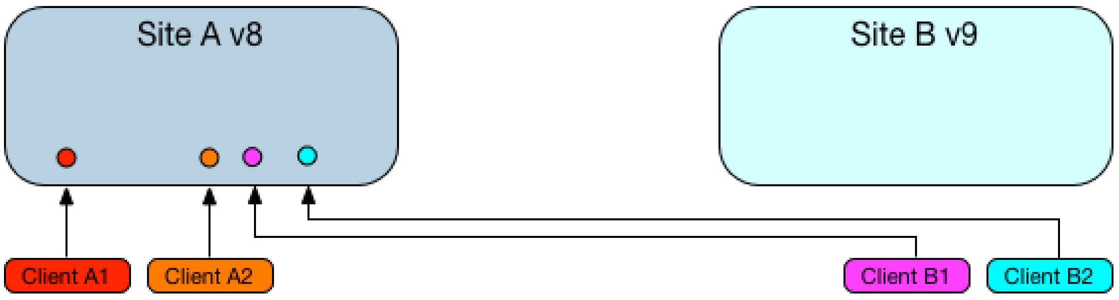
3. Upgrade Site B to GemFire 9.x.



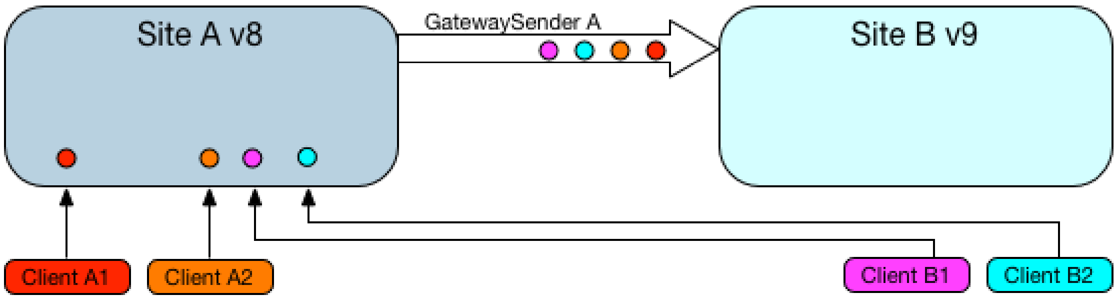
4. Start GemFire 9.x on Site B.



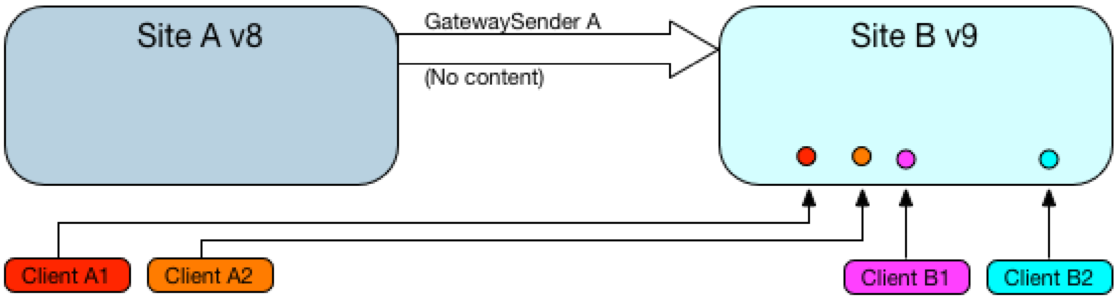
5. Pause the gateway senders on Site B.



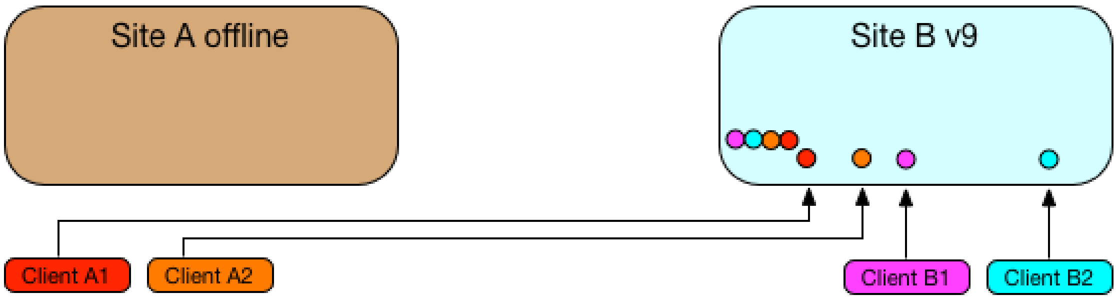
6. Start the gateway senders on Site A and wait for Site A's queues to drain.



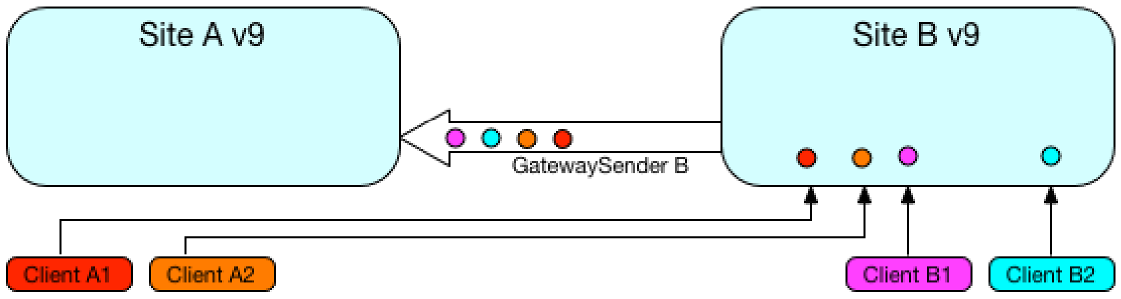
7. Once the queues have drained sufficiently, reconfigure to redirect all traffic to Site B. Site A's queues should now drain the rest of the way.



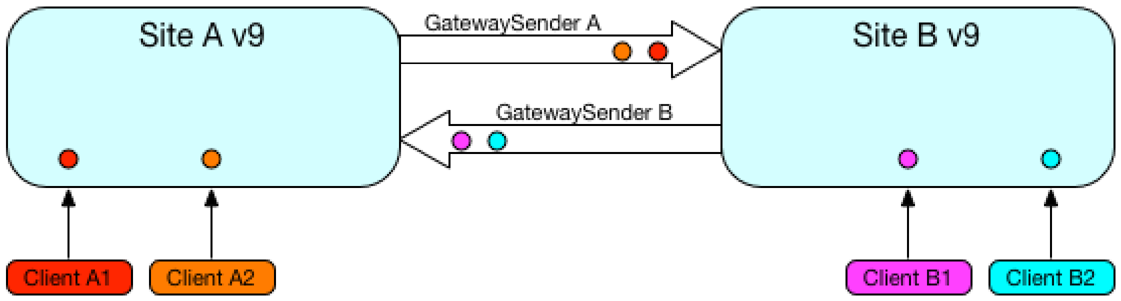
8. Upgrade Site A to GemFire 9.x.



9. Once Site A is back up, start Site B's gateway senders and wait for the queues to drain sufficiently.



10. Reconfigure to redistribute the load across both Site A and Site B.



Pivotal GemFire in 15 Minutes or Less

Need a quick introduction to Pivotal GemFire? Take this brief tour to try out basic features and functionality.

Step 1. Install Pivotal GemFire

See Installing Pivotal GemFire for instructions.

Step 2. Use gfsh to start a locator

In a terminal window, use the `gfsh` command line interface to start up a locator. *gfsh* (pronounced “jee-fish”) provides a single, intuitive command-line interface from which you can launch, manage, and monitor Pivotal GemFire processes, data, and applications. See gfsh.

The *locator* is a GemFire process that tells new, connecting members where running members are located and provides load balancing for server use. A locator, by default, starts up a JMX Manager, which is used for monitoring and managing a GemFire cluster. The cluster configuration service uses locators to persist and distribute cluster configurations to cluster members. See Running GemFire Locator Processes and Overview of the Cluster Configuration Service.

1. Create a scratch working directory (for example, `my_gemfire`) and change directories into it. `gfsh` saves locator and server working directories and log files in this location.
2. Start gfsh by typing `gfsh` at the command line (or `gfsh.bat` if you are using Windows).

```
    _________________________     __
   / _____/ ______/ ______/ /____/ /
  / /  __/ /___  /_____  / _____  /
 / /__/ / ____/  _____/ / /    / /
/______/_/      /______/_/    /_/
```

```
Monitor and Manage GemFire
gfsh>
```

3. At the `gfsh` prompt, type the `start locator` command and specify a name for the locator:

```
gfsh>start locator --name=locator1
Starting a GemFire Locator in /home/username/my_gemfire/locator1...
.................................
Locator in /home/username/my_gemfire/locator1 on ubuntu.local[10334] as locator
1 is currently online.
Process ID: 3529
Uptime: 18 seconds
GemFire Version: 9.0.0
Java Version: 1.8.0_272
Log File: /home/username/my_gemfire/locator1/locator1.log
JVM Arguments: -Dgemfire.enable-cluster-configuration=true -Dgemfire.load-clust
er-configuration-from-dir=false
-Dgemfire.launcher.registerSignalHandlers=true -Djava.awt.headless=true
-Dsun.rmi.dgc.server.gcInterval=9223372036854775806
Class-Path: /home/username/pivotal_gemfire-9.0.0/lib/geode-dependencies.jar

Successfully connected to: [host=ubuntu.local, port=1099]

Cluster configuration service is up and running.
```

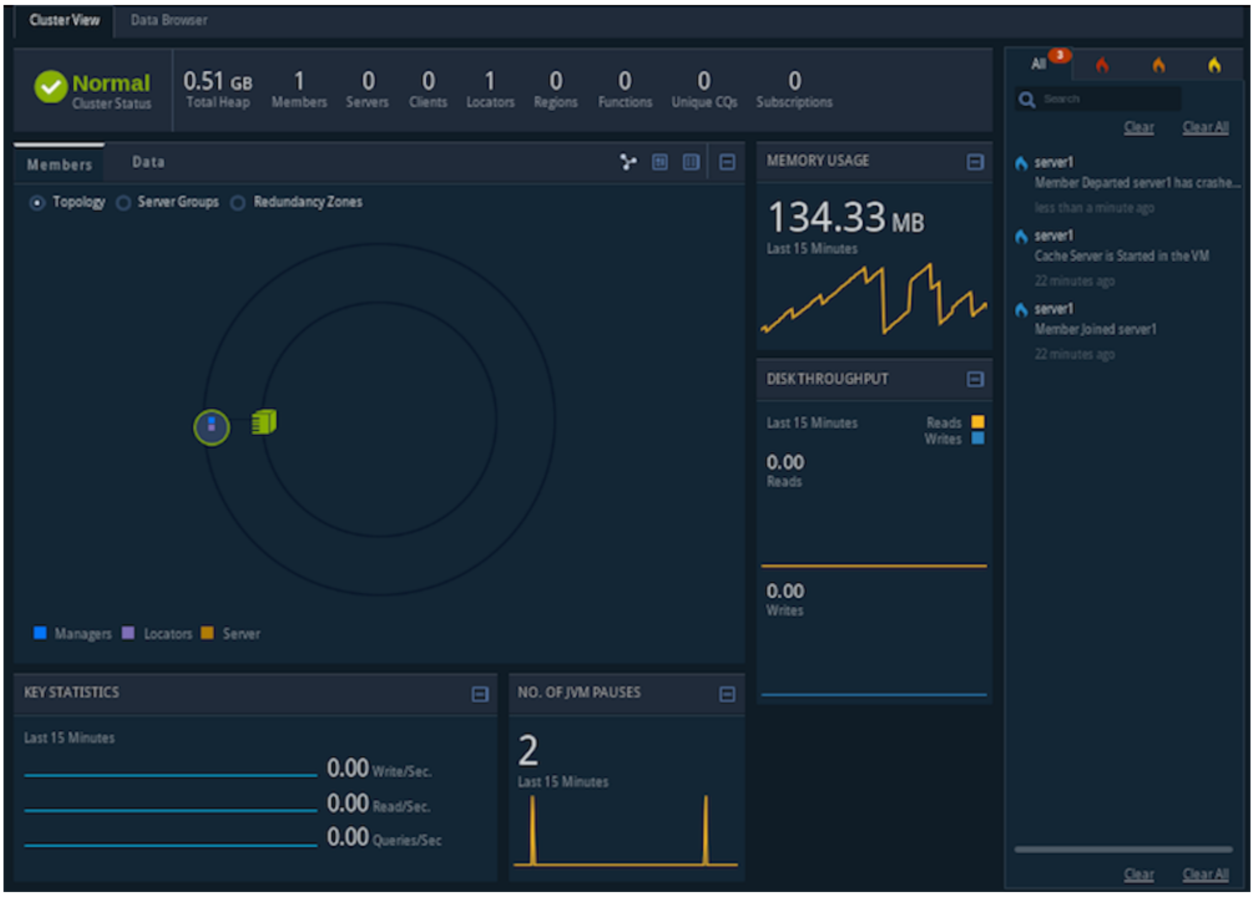
Step 3. Start GemFire Pulse

Start up the browser-based GemFire Pulse monitoring tool. GemFire Pulse is a Web Application that provides a graphical dashboard for monitoring vital, real-time health and performance of GemFire clusters, members, and regions. See GemFire Pulse.

```
gfsh>start pulse
Launched GemFire Pulse
```

>This command launches Pulse and automatically connects you to the JMX Manager running in the Locator. At the Pulse login screen, type in the default username `admin` and password `admin`.

The Pulse application now displays the locator you just started (locator1):



server1
Member Departed server1 has crashe...
less than a minute ago

server1
Cache Server is Started in the VM
22 minutes ago

server1
Member Joined server1
22 minutes ago

Step 4. Start a server

A GemFire server is a Pivotal GemFire process that runs as a long-lived, configurable member of a cluster (also called a *distributed system*). The GemFire server is used primarily for hosting long-lived data regions and for running standard GemFire processes such as the server in a client/server configuration. See Running GemFire Server Processes.

Start the cache server:

```
gfsh>start server --name=server1 --server-port=40411
```

This commands starts a cache server named “server1” on the specified port of 40411.

Observe the changes (new member and server) in Pulse. Try expanding the distributed system icon to see the locator and cache server graphically.

Step 5. Create a replicated, persistent region

In this step you create a region with the `gfsh` command line utility. Regions are the core building blocks of the GemFire cluster and provide the means for organizing your data. The region you create for this exercise employs replication to replicate data across members of the cluster and utilizes persistence to save the data to disk. See Data Regions.

1. Create a replicated, persistent region:

```
gfsh>create region --name=regionA --type=REPLICATE_PERSISTENT
Member   | Status
```

```
------- | --------------------------------------
server1 | Region "/regionA" created on "server1"
```

Note that the region is hosted on server1.

2. Use the `gfsh` command line to view a list of the regions in the cluster.

```
gfsh>list regions
List of regions
---------------
regionA
```

3. List the members of your cluster. The locator and cache server you started appear in the list:

```
gfsh>list members
  Name    | Id
-------- | ---------------------------------------
locator1 | 192.0.2.0(locator1:3529:locator)<v0>:59926
server1  | 192.0.2.0(server1:3883)<v1>:65390
```

4. To view specifics about a region, type the following:

```
gfsh>describe region --name=regionA
..........................................................
Name            : regionA
Data Policy     : persistent replicate
Hosting Members : server1

Non-Default Attributes Shared By Hosting Members

 Type  | Name | Value
------ | ---- | -----
Region | size | 0
```

5. In Pulse, click the green cluster icon to see all the new members and new regions that you just added to your cluster.

Note: Keep this `gfsh` prompt open for the next steps.

Step 6. Manipulate data in the region and demonstrate persistence

Pivotal GemFire manages data as key/value pairs. In most applications, a Java program adds, deletes and modifies stored data. You can also use gfsh commands to add and retrieve data. See Basic GemFire gfsh Commands.

1. Run the following `put` commands to add some data to the region:

```
gfsh>put --region=regionA --key="1" --value="one"
Result      : true
Key Class   : java.lang.String
Key         : 1
Value Class : java.lang.String
Old Value   : <NULL>
```

```
gfsh>put --region=regionA --key="2" --value="two"
Result      : true
Key Class   : java.lang.String
Key         : 2
Value Class : java.lang.String
Old Value   : <NULL>
```

2. Run the following command to retrieve data from the region:

```
gfsh>query --query="select * from /regionA"

Result     : true
startCount : 0
endCount   : 20
Rows       : 2

Result
------
two
one
```

   Note that the result displays the values for the two data entries you created with the `put` commands.

   See Data Entries.

3. Stop the cache server using the following command:

```
gfsh>stop server --name=server1
Stopping Cache Server running in /home/username/my_gemfire/server1 on ubuntu.lo
cal[40411] as server1...
Process ID: 3883
Log File: /home/username/my_gemfire/server1/server1.log
....
```

4. Restart the cache server using the following command:

```
gfsh>start server --name=server1 --server-port=40411
```

5. Run the following command to retrieve data from the region again – notice that the data is still available:

```
gfsh>query --query="select * from /regionA"

Result     : true
startCount : 0
endCount   : 20
Rows       : 2

Result
------
two
one
```

Because regionA uses persistence, it writes a copy of the data to disk. When a server hosting regionA starts, the data is populated into the cache. Note that the result displays the values for the two data entries you created with the `put` commands prior to stopping the server.

See Data Entries.

See Data Regions.

Step 7. Examine the effects of replication

In this step, you start a second cache server. Because regionA is replicated, the data will be available on any server hosting the region.

1. Start a second cache server:

```
gfsh>start server --name=server2 --server-port=40412
```

2. Run the `describe region` command to view information about regionA:

```
gfsh>describe region --name=regionA
..........................................................
Name            : regionA
Data Policy     : persistent replicate
Hosting Members : server1
                  server2

Non-Default Attributes Shared By Hosting Members

 Type  | Name | Value
------ | ---- | -----
Region | size | 2
```

Note that you do not need to create regionA again for server2. The output of the command shows that regionA is hosted on both server1 and server2. When gfsh starts a server, it requests the configuration from the cluster configuration service which then distributes the shared configuration to any new servers joining the cluster.

3. Add a third data entry:

```
gfsh>put --region=regionA --key="3" --value="three"
Result      : true
Key Class   : java.lang.String
Key         : 3
Value Class : java.lang.String
Old Value   : <NULL>
```

4. Open the Pulse application (in a Web browser) and observe the cluster topology. You should see a locator with two attached servers. Click the **Data** tab to view information about regionA.

5. Stop the first cache server with the following command:

```
gfsh>stop server --name=server1
Stopping Cache Server running in /home/username/my_gemfire/server1 on ubuntu.lo
```

```
cal[40411] as server1...
Process ID: 4064
Log File: /home/username/my_gemfire/server1/server1.log
....
```

6. Retrieve data from the remaining cache server.

```
gfsh>query --query="select * from /regionA"

Result     : true
startCount : 0
endCount   : 20
Rows       : 3

Result
------
two
one
three
```

   Note that the data contains 3 entries, including the entry you just added.

7. Add a fourth data entry:

```
gfsh>put --region=regionA --key="4" --value="four"
Result      : true
Key Class   : java.lang.String
Key         : 3
Value Class : java.lang.String
Old Value   : <NULL>
```

   Note that only server2 is running. Because the data is replicated and persisted, all of the data is still available. But the new data entry is currently only available on server 2.

```
gfsh>describe region --name=regionA
..........................................................
Name             : regionA
Data Policy      : persistent replicate
Hosting Members  : server2

Non-Default Attributes Shared By Hosting Members

 Type  | Name | Value
------ | ---- | -----
Region | size | 4
```

8. Stop the remaining cache server:

```
gfsh>stop server --name=server2
Stopping Cache Server running in /home/username/my_gemfire/server2 on ubuntu.lo
cal[40412] as server2...
Process ID: 4185
Log File: /home/username/my_gemfire/server2/server2.log
.....
```

Step 8. Restart the cache servers in parallel

In this step you restart the cache servers in parallel. Because the data is persisted, the data is available when the servers restart. Because the data is replicated, you must start the servers in parallel so that they can synchronize their data before starting.

1. Start server1. Because regionA is replicated and persistent, it needs data from the other server to start and waits for the server to start:

   ```
   gfsh>start server --name=server1 --server-port=40411
   Starting a GemFire Server in /home/username/my_gemfire/server1...
   ............................................................................
   ............................................................................
   ```

   Note that if you look in the `server1.log` file for the restarted server, you will see a log message similar to the following:

   ```
   [info 2015/01/14 09:08:13.610 PST server1 <main> tid=0x1] Region /regionA has p
   ot
   entially stale data. It is waiting for another member to recover the latest dat
   a.
     My persistent id:

       DiskStore ID: 8e2d99a9-4725-47e6-800d-28a26e1d59b1
       Name: server1
       Location: /192.0.2.0:/home/username/my_gemfire/server1/.

     Members with potentially new data:
     [
       DiskStore ID: 2e91b003-8954-43f9-8ba9-3c5b0cdd4dfa
       Name: server2
       Location: /192.0.2.0:/home/username/my_gemfire/server2/.
     ]
     Use the "gemfire list-missing-disk-stores" command to see all disk stores tha
   t
   are being waited on by other members.
   ```

2. In a second terminal window, change directories to the scratch working directory (for example, `my_gemfire`) and start gfsh:

   ```
   [username@localhost ~/my_gemfire]$ gfsh
       _________________________     __
      / _____/ ______/ ______/ /____/ /
     / /  __/ /___  /_____  / _____  /
    / /__/ / ____/  _____/ / /    / /
   /______/_/      /______/_/    /_/

   Monitor and Manage GemFire
   ```

3. Run the following command to connect to the cluster:

   ```
   gfsh>connect --locator=localhost[10334]
   Connecting to Locator at [host=localhost, port=10334] ..
   Connecting to Manager at [host=ubuntu.local, port=1099] ..
   Successfully connected to: [host=ubuntu.local, port=1099]
   ```

4. Start server2:

```
gfsh>start server --name=server2 --server-port=40412
```

When server2 starts, note that **server1 completes its start up** in the first gfsh window:

```
Server in /home/username/my_gemfire/server1 on ubuntu.local[40411] as server1 i
s currently online.
Process ID: 3402
Uptime: 1 minute 46 seconds
GemFire Version: 9.0.0
Java Version: 1.8.0_272
Log File: /home/username/my_gemfire/server1/server1.log
JVM Arguments: -Dgemfire.default.locators=192.0.2.0[10334] -Dgemfire.use-cluste
r-configuration=true
-XX:OnOutOfMemoryError=kill -KILL %p -Dgemfire.launcher.registerSignalHandlers=
true
-Djava.awt.headless=true -Dsun.rmi.dgc.server.gcInterval=9223372036854775806
Class-Path: /home/username/pivotal_gemfire-9.0.0/lib/geode-dependencies.jar
```

5. Verify that the locator and two servers are running:

```
gfsh>list members
  Name    | Id
--------- | ---------------------------------------
server2   | ubuntu(server2:3992)<v8>:21507
server1   | ubuntu(server1:3402)<v7>:36532
locator1  | ubuntu(locator1:2813:locator)<v0>:46644
```

6. Run a query to verify that all the data you entered with the `put` commands is available:

```
gfsh>query --query="select * from /regionA"

Result     : true
startCount : 0
endCount   : 20
Rows       : 5

Result
------
one
two
four
Three

NEXT_STEP_NAME : END
```

7. Stop server2 with the following command:

```
gfsh>stop server --dir=server2
Stopping Cache Server running in /home/username/my_gemfire/server2 on 192.0.2.0
[40412] as server2...
Process ID: 3992
Log File: /home/username/my_gemfire/server2/server2.log
....
```

8. Run a query to verify that all the data you entered with the `put` commands is still available:

```
gfsh>query --query="select * from /regionA"

Result      : true
startCount  : 0
endCount    : 20
Rows        : 5

Result
------
one
two
four
Three

NEXT_STEP_NAME : END
```

Step 9. Shut down the system including your locators

To shut down your cluster, do the following:

1. In the current `gfsh` session, stop the cluster:

   ```
   gfsh>shutdown --include-locators=true
   ```

   See the shutdown manual page.

2. When prompted, type ‘Y’ to confirm the shutdown of the cluster.

   ```
   As a lot of data in memory will be lost, including possibly events in queues,
   do you really want to shutdown the entire distributed system? (Y/n): Y
   Shutdown is triggered

   gfsh>
   No longer connected to ubuntu.local[1099].
   gfsh>
   ```

3. Type `exit` to quit the gfsh shell.

Step 10. What to do next...

Here are some suggestions on what to explore next with Pivotal GemFire:

- Continue reading the next section to learn more about the components and concepts that were just introduced.
- To get more practice using `gfsh`, see Tutorial—Performing Common Tasks with gfsh.
- To learn about the cluster configuration service, see Tutorial—Creating and Using a Cluster Configuration.

Configuring and Running a Cluster

Use the `gfsh` command-line utility to configure your VMware GemFire cluster. The cluster configuration service persists the cluster configurations and distributes the configurations to members of the cluster. There are also several additional ways to configure a cluster.

Use `gfsh` to configure regions, disk stores, members, and other VMware GemFire objects. You can also use `gfsh` to start and stop locators, servers, and VMware GemFire monitoring tools. As you execute these commands, the cluster configuration service persists the configuration. When new members join the cluster, the service distributes the configuration to the new members.

`gfsh` is the recommended means of configuring and managing your VMware GemFire cluster, however you can still configure many aspects of a cluster using the older methods of the cache.xml and gemfire.properties files. See cache.xml and the Reference for configuration parameters. You can also configure some aspects of a cluster using a Java API. See Managing VMware GemFire.

- **Overview of the Cluster Configuration Service**

  The VMware GemFire cluster configuration service persists cluster configurations created by `gfsh` commands to the locators in a cluster and distributes the configurations to members of the cluster.

- **Tutorial—Creating and Using a Cluster Configuration**

  A short walk-through that uses a single computer to demonstrate how to use `gfsh` to create a cluster configuration for a VMware GemFire cluster.

- **Deploying Application JARs to VMware GemFire Members**

  You can dynamically deploy your application JAR files to specific members or to all members in your cluster. VMware GemFire automatically keeps track of JAR file versions; autoloads the deployed JAR files to the CLASSPATH; and auto-registers any functions that the JAR contains.

- **Using Member Groups**

  VMware GemFire allows you to organize your cluster members into logical member groups.

- **Exporting and Importing Cluster Configurations**

  The cluster configuration service exports and imports configurations created using `gfsh` for an entire VMware GemFire cluster.

- **Cluster Configuration Files and Troubleshooting**

  When you use the cluster configuration service in VMware GemFire, you can examine the generated configuration files in the `cluster_config` directory on the locator. `gfsh` saves configuration files at the cluster-level and at the individual group-level.

- **Using gfsh to Manage a Remote Cluster Over HTTP or HTTPS**

You can connect `gfsh` via HTTP or HTTPS to a remote cluster and manage the cluster using `gfsh` commands.

- **Deploying Configuration Files without the Cluster Configuration Service**

  You can deploy your VMware GemFire configuration files in your system directory structure or in jar files. You determine how you want to deploy your configuration files and set them up accordingly.

- **Starting Up and Shutting Down Your System**

  Determine the proper startup and shutdown procedures, and write your startup and shutdown scripts.

- **Running VMware GemFire Locator Processes**

  The locator is a VMware GemFire process that tells new, connecting members where running members are located and provides load balancing for server use.

- **Running VMware GemFire Server Processes**

  A VMware GemFire server is a process that runs as a long-lived, configurable member of a client/server system.

- **Managing System Output Files**

  VMware GemFire output files are optional and can become quite large. Work with your system administrator to determine where to place them to avoid interfering with other system activities.

- **Firewall Considerations**

  You can configure and limit port usage for situations that involve firewalls, for example, between client-server or server-server connections.

Overview of the Cluster Configuration Service

The VMware GemFire cluster configuration service persists cluster configurations created by `gfsh` commands to the locators in a cluster and distributes the configurations to members of the cluster.

Why Use the Cluster Configuration Service

We highly recommend that you use the `gfsh` command line and the cluster configuration service as the primary mechanism to manage your cluster configuration. Specify configuration within a `cache.xml` file for only those items that cannot be specified or altered using `gfsh`. Using a common cluster configuration reduces the amount of time you spend configuring individual members and enforces consistent configurations when bringing up new members in your cluster. You no longer need to reconfigure each new member that you add to the cluster. You no longer need to worry about validating your `cache.xml` file. It also becomes easier to propagate configuration changes across your cluster and deploy your configuration changes to different environments.

You can use the cluster configuration service to:

- Save the configuration for an entire VMware GemFire cluster.
- Restart members using a previously-saved configuration.

- Export a configuration from a development environment and migrate that configuration to create a testing or production system.
- Start additional servers without having to configure each server separately.
- Configure some servers to host certain regions and other servers to host different regions, and configure all servers to host a set of common regions.

Using the Cluster Configuration Service

To use the cluster configuration service in VMware GemFire, you must use dedicated, standalone locators in your deployment. You cannot use the cluster configuration service with co-located locators (locators running in another process such as a server) or in multicast environments.

The standalone locators distribute configuration to all locators in a cluster. Every locator in the cluster with `--enable-cluster-configuration` set to true keeps a record of all cluster-level and group-level configuration settings.

Note: The default behavior for `gfsh` is to create and save cluster configurations. You can deactivate the cluster configuration service by using the `--enable-cluster-configuration=false` option when starting locators.

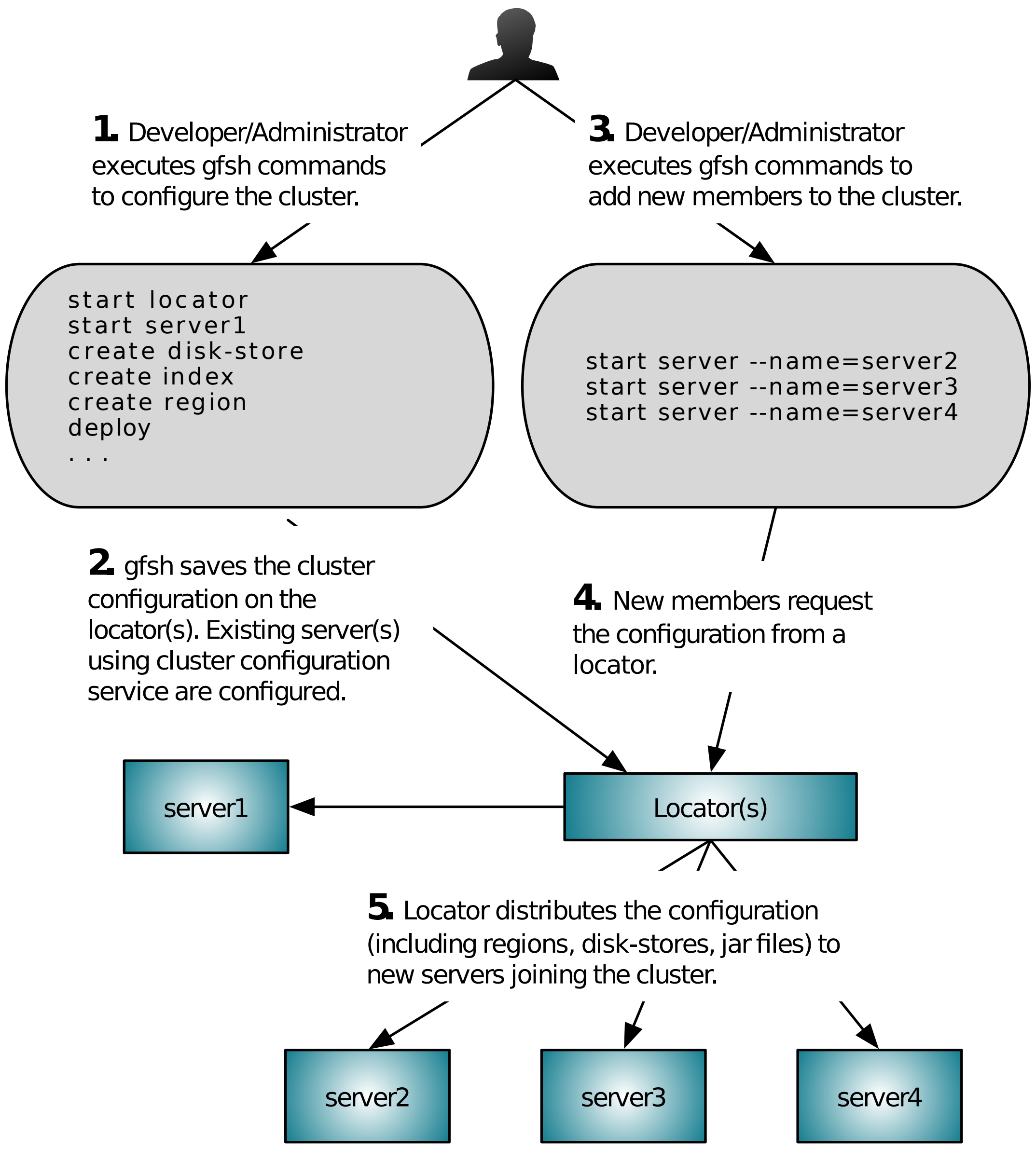
You can load existing configuration into the cluster by using the `gfsh import cluster-configuration` command after starting up a locator.

Subsequently, any servers that you start with `gfsh` that have `--use-cluster-configuration` set to `true` will pick up the cluster configuration from the locator as well as any appropriate group-level configurations (for member groups they belong to). To deactivate the cluster configuration service on a server, you must start the server with the `--use-cluster-configuration` parameter set to `false`. By default, the parameter is set to true.

How the Cluster Configuration Service Works

When you use `gfsh` commands to create VMware GemFire regions, disk-stores, and other objects, the cluster configuration service saves the configurations on each locator in the cluster. If you specify a group when issuing these commands, a separate configuration is saved containing only configurations that apply to the group.

When you use `gfsh` to start new VMware GemFire servers, the locator distributes the persisted configurations to the new server. If you specify a group when starting the server, the server receives the group-level configuration in addition to the cluster-level configuration. Group-level configurations are applied after cluster-wide configurations; therefore you can use group-level to override cluster-level settings.


1. Developer/Administrator executes gfsh commands to configure the cluster.
start locator
start server1
create disk-store
create index
create region
deploy
. . .
3. Developer/Administrator executes gfsh commands to add new members to the cluster.
start server --name=server2
start server --name=server3
start server --name=server4
2. gfsh saves the cluster configuration on the locator(s). Existing server(s) using cluster configuration service are configured.
4. New members request the configuration from a locator.
server1
Locator(s)
5. Locator distributes the configuration (including regions, disk-stores, jar files) to new servers joining the cluster.
server2
server3
server4

1. Developer or Administrator executes `gfsh` commands to configure the cluster.
2. `gfsh` saves the cluster information on the locators. Existing servers using cluster configuration are configured.
3. Developer or Administrator executes `gfsh` to add new members to the cluster.
4. New members request the configuration from a locator.
5. Locator distributes the configuration to new servers joining the cluster. Configuration includes regions, disk stores, and jar files,

gfsh Commands that Create Cluster Configurations

The following `gfsh` commands cause the configuration to be written to all locators in the cluster (the locators write the configuration to disk):

- `configure pdx`*

- `create region`
- `alter region`
- `alter runtime`
- `destroy region`
- `create index`
- `destroy index`
- `create disk-store`
- `destroy disk-store`
- `create async-event-queue`
- `alter async-event-queue`
- `destroy async-event-queue`
- `deploy jar`
- `undeploy jar`
- `create gateway-sender`
- `destroy gateway-sender`
- `create gateway-receiver`
- `destroy gateway-receiver`
- `alter query-service`

* Note that the `configure pdx` command must be executed *before* starting your data members. This command does not affect any currently running members in the system. Data members (with cluster configuration enabled) that are started after running this command will pick up the new PDX configuration.

gfsh Limitations

These are the configurations that you cannot create or alter using `gfsh`. These configurations must be within a `cache.xml` file or be applied by using the API:

- Client cache configuration
- You cannot directly modify the attributes of the following objects:
    - `function`
    - `custom-load-probe`
    - `compressor`
    - `serializer`
    - `instantiator`
    - `pdx-serializer`

      Note: The `configure pdx` command always specifies the `org.apache.geode.pdx.ReflectionBasedAutoSerializer` class. You cannot specify a custom PDX serializer in gfsh.

  - `initializer`
  - `lru-heap-percentage`
  - `lru-memory-size`
  - `partition-resolver`
  - `partition-listener`
  - `transaction-listener`
  - `transaction-writer`

- Adding or removing a `TransactionListener`
- Configuring a `GatewayConflictResolver`
- You cannot specify parameters and values for Java classes for the following:
  - `gateway-listener`
  - `gateway-conflict-resolver`
  - `gateway-event-filter`
  - `gateway-transport-filter`
  - `gateway-event-substitution-filter`

Deactivating the Cluster Configuration Service

If you do not want to use the cluster configuration service, start up your locator with the `--enable-cluster-configuration` parameter set to false or do not use standalone locators. You will then need to configure the cache (via cache.xml or API) separately on all your cluster members.

Tutorial—Creating and Using a Cluster Configuration

A short walk-through that uses a single computer to demonstrate how to use `gfsh` to create a cluster configuration for a VMware GemFire cluster.

The `gfsh` command-line tool allows you to configure and start a VMware GemFire cluster. The cluster configuration service uses VMware GemFire locators to store the configuration at the group and cluster levels and serves these configurations to new members as they are started. The locators store the configurations in a hidden region that is available to all locators and also write the configuration data to disk as XML files. Configuration data is updated as `gfsh` commands are executed.

This section provides a walk-through example of configuring a simple VMware GemFire cluster and then re-using that configuration in a new context.

1. Create a working directory (For example:`/home/username/my_geode`) and switch to the new directory. This directory will contain the configurations for your cluster.
2. Start the `gfsh` command-line tool. For example:

   ```
   $ gfsh
   ```

The `gfsh` command prompt displays.

```
    _________________________     __
   / _____/ ______/ ______/ /____/ /
  / /  __/ /___  /_____  / _____  /
 / /__/ / ____/  _____/ / /    / /
/______/_/      /______/_/    /_/    9.10

Monitor and Manage VMware GemFire
gfsh>
```

3. Start a locator using the command in the following example:

```
gfsh>start locator --name=locator1
Starting a VMware GemFire Locator in /Users/username/my_geode/locator1...
.............................
Locator in /Users/username/my_geode/locator1 on 192.0.2.0[10334] as locator1
  is currently online.
Process ID: 70919
Uptime: 12 seconds
VMware GemFire Version: 9.10
Java Version: 1..0_272
Log File: /Users/username/my_geode/locator1/locator1.log
JVM Arguments: -Dgemfire.enable-cluster-configuration=true
-Dgemfire.load-cluster-configuration-from-dir=false
-Dgemfire.launcher.registerSignalHandlers=true -Djava.awt.headless=true
-Dsun.rmi.dgc.server.gcInterval=9223372036854775806
Class-Path: /Users/username/geode910/lib/geode-dependencies.jar

Successfully connected to: JMX Manager [host=192.0.2.0, port=1099]

Cluster configuration service is up and running.
```

Note that `gfsh` responds with a message indicating that the cluster configuration service is up and running. If you see a message indicating a problem, review the locator log file for possible errors. The path to the log file is displayed in the output from `gfsh`.

4. Start VMware GemFire servers using the commands in the following example:

```
gfsh>start server --name=server1 --groups=group1
Starting a VMware GemFire Server in /Users/username/my_geode/server1...
.....
Server in /Users/username/my_geode/server1 on 192.0.2.0[40404] as server1
  is currently online.
Process ID: 5627
Uptime: 2 seconds
VMware GemFire Version: 9.10
Java Version: 1..0_272
Log File: /Users/username/my_geode/server1/server1.log

JVM Arguments: -Dgemfire.default.locators=192.0.2.0[10334] -Dgemfire.groups=gro
up1
-Dgemfire.start-dev-rest-api=false -Dgemfire.use-cluster-configuration=true
-Dgemfire.launcher.registerSignalHandlers=true -Djava.awt.headless=true
Class-Path: /Users/username/geode910/lib/geode-dependencies.jar

gfsh>start server --name=server2 --groups=group1 --server-port=40405
```

```
Starting a VMware GemFire Server in /Users/username/my_geode/server2...
.....
Server in /Users/username/my_geode/server2 on 192.0.2.0[40405] as server2
  is currently online.
Process ID: 5634
Uptime: 2 seconds
VMware GemFire Version: 9.10
Java Version: 1..0_272
Log File: /Users/username/my_geode/server2/server2.log

JVM Arguments: -Dgemfire.default.locators=192.0.2.0[10334] -Dgemfire.groups=gro
up1
-Dgemfire.start-dev-rest-api=false -Dgemfire.use-cluster-configuration=true
-Dgemfire.launcher.registerSignalHandlers=true -Djava.awt.headless=true
-Dsun.rmi.dgc.server.gcInterval=9223372036854775806
Class-Path: /Users/username/geode910/lib/geode-dependencies.jar

gfsh>start server --name=server3 --server-port=40406
Starting a VMware GemFire Server in /Users/username/my_geode/server3...
.....
Server in /Users/username/my_geode/server3 on 192.0.2.0[40406] as server3
  is currently online.
Process ID: 5637
Uptime: 2 seconds
VMware GemFire Version: 9.10
Java Version: 1..0_272
Log File: /Users/username/my_geode/server3/server3.log
JVM Arguments: -Dgemfire.default.locators=192.0.2.0[10334]
-Dgemfire.start-dev-rest-api=false -Dgemfire.use-cluster-configuration=true
-Dgemfire.launcher.registerSignalHandlers=true -Djava.awt.headless=true
-Dsun.rmi.dgc.server.gcInterval=9223372036854775806
Class-Path: /Users/username/geode910/lib/geode-dependencies.jar
```

Note that the `gfsh` commands you used to start `server1` and `server2` specify a group named `group1` while the command for `server3` did not specify a group name.

5. Create some regions using the commands in the following example:

```
gfsh>create region --name=region1 --groups=group1 --type=REPLICATE
Member  | Status | Message
------- | ------ | --------------------------------------
server1 | OK     | Region "/region1" created on "server1"
server2 | OK     | Region "/region1" created on "server2"

Cluster configuration for group 'group1' is updated.


gfsh>create region --name=region2 --type=REPLICATE
Member  | Status | Message
------- | ------ | --------------------------------------
server1 | OK     | Region "/region2" created on "server1"
server2 | OK     | Region "/region2" created on "server2"
server3 | OK     | Region "/region2" created on "server3"

Cluster configuration for group 'cluster' is updated.
```

Note that `region1` is created on all cache servers that specified the group named `group1` when starting the cache server (`server1` and `server2`, in this example). `region2` is created on all members because no group was specified.

6. Deploy jar files. Use the `gfsh deploy` command to deploy application jar files to all members or to a specified group of members. The following example deploys the `mx4j-3.0.2.jar` and `ra.jar` files from the distribution. (Note: This is only an example, you do not need to deploy these files to use the Cluster Configuration Service. Alternately, you can use any two jar files for this demonstration.)

```
gfsh>deploy --groups=group1 --jars=/lib/mx4j-3.0.2.jar

Deploying files: mx4j-3.0.2.jar
Total file size is: 0.39MB

Continue?  (Y/n): y
Member   |  Deployed JAR  | Deployed JAR Location
------- | -------------- | ----------------------------------------------------
------
server1 | mx4j-3.0.2.jar | /Users/username/my_geode/server1/mx4j-3.0.2.v1.jar
server2 | mx4j-3.0.2.jar | /Users/username/my_geode/server2/mx4j-3.0.2.v1.jar

gfsh>deploy --jars=/lib/ra.jar

Deploying files: ra.jar
Total file size is: 0.03MB

Continue?  (Y/n): y
Member  | Deployed JAR | Deployed JAR Location
------- | ------------ | ---------------------------------------------------
server1 | ra.jar       | /Users/username/my_geode/server1/ra.v1.jar
server2 | ra.jar       | /Users/username/my_geode/server2/ra.v1.jar
server3 | ra.jar       | /Users/username/my_geode/server2/ra.v1.jar
```

Note that the `mx4j-3.0.2.jar` file was deployed only to the members of `group1` and the `ra.jar` was deployed to all members.

7. Export the cluster configuration. You can use the `gfsh export cluster-configuration` command to create a zip file that contains the cluster's persisted configuration. The zip file contains a copy of the contents of the `cluster_config` directory. For example:

```
gfsh>export cluster-configuration --zip-file-name=/Users/username/myClConfig.zi
p
```

VMware GemFire writes the cluster configuration to the specified zip file.

```
File saved to /Users/username/myClConfig.zip
```

The remaining steps demonstrate how to use the cluster configuration you just created.

8. Shut down the cluster using the following commands:

```
gfsh>shutdown --include-locators=true
As a lot of data in memory will be lost, including possibly events in queues, d
o you
really want to shutdown the entire distributed system? (Y/n): Y
```

```
Shutdown is triggered

gfsh>
No longer connected to 192.0.2.0[1099].
gfsh>
```

9. Exit the `gfsh` command shell:

```
gfsh>quit
Exiting...
```

10. Create a new working directory (for example: `new_geode`) and switch to the new directory.
11. Start the `gfsh` command shell:

```
$ gfsh
```

12. Start a new locator. For example:

```
gfsh>start locator --name=locator2 --port=10335
Starting a VMware GemFire Locator in /Users/username/new_geode/locator2...
.............................
Locator in /Users/username/new_geode/locator2 on 192.0.2.0[10335] as locator2
  is currently online.
Process ID: 5749
Uptime: 15 seconds
VMware GemFire Version: 9.10
Java Version: 1..0_272
Log File: /Users/username/new_geode/locator2/locator2.log

JVM Arguments: -Dgemfire.enable-cluster-configuration=true
-Dgemfire.load-cluster-configuration-from-dir=false
-Dgemfire.launcher.registerSignalHandlers=true -Djava.awt.headless=true
-Dsun.rmi.dgc.server.gcInterval=9223372036854775806
Class-Path: /Users/username/geode910/lib/geode-dependencies.jar

Successfully connected to: JMX Manager [host=192.0.2.0, port=1099]

Cluster configuration service is up and running.
```

13. Import the cluster configuration using the `import cluster-configuration` command. For example:

```
gfsh>import cluster-configuration --zip-file-name=/Users/username/myClConfig.zi
p
This command will replace the existing cluster configuration, if any, The old c
onfiguration will be backed up in the working directory.

Continue?  (Y/n): y
Cluster configuration successfully imported
```

Note that the `locator2` directory now contains a `cluster_config` subdirectory.

14. Start a server that does not reference a group:

```
gfsh>start server --name=server4 --server-port=40414
Starting a VMware GemFire Server in /Users/username/new_geode/server4...
```

```
........
Server in /Users/username/new_geode/server4 on 192.0.2.0[40414] as server4
is currently online.
Process ID: 5813
Uptime: 4 seconds
VMware GemFire Version: 9.10
Java Version: 1..0_272
Log File: /Users/username/new_geode/server4/server4.log

JVM Arguments: -Dgemfire.default.locators=192.0.2.0[10335]
-Dgemfire.start-dev-rest-api=false -Dgemfire.use-cluster-configuration=true
-Dgemfire.launcher.registerSignalHandlers=true -Djava.awt.headless=true
-Dsun.rmi.dgc.server.gcInterval=9223372036854775806
Class-Path: /Users/username/geode910/lib/geode-dependencies.jar
```

15. Start another server that references `group1`:

```
gfsh>start server --name=server5 --groups=group1 --server-port=40415
Starting a VMware GemFire Server in /Users/username/new_geode/server5...
.....
Server in /Users/username/new_geode/server2 on 192.0.2.0[40415] as server5
is currently online.
Process ID: 5954
Uptime: 2 seconds
VMware GemFire Version: 9.10
Java Version: 1..0_272
Log File: /Users/username/new_geode/server5/server5.log
JVM Arguments: -Dgemfire.default.locators=192.0.2.0[10335] -Dgemfire.groups=gro
up1
-Dgemfire.start-dev-rest-api=false -Dgemfire.use-cluster-configuration=true
-Dgemfire.launcher.registerSignalHandlers=true -Djava.awt.headless=true
-Dsun.rmi.dgc.server.gcInterval=9223372036854775806
Class-Path: /Users/username/geode910/lib/geode-dependencies.jar
```

16. Use the `list regions` command to display the configured regions. Note that region1 and region2, which were configured in the original cluster level are available.

```
gfsh>list regions
List of regions
---------------
region1
region2
```

17. Use the `describe region` command to see which members host each region. Note that region1 is hosted only by server5 because server5 was started using the group1 configuration. region2 is hosted on both server4 and server5 because region2 was created without a group specified.

```
gfsh>describe region --name=region1
Name            : region1
Data Policy     : replicate
Hosting Members : server5

Non-Default Attributes Shared By Hosting Members

 Type  |    Name     | Value
------ | ----------- | ---------------
```

```
Region | data-policy | REPLICATE
       | size        | 0
       | scope       | distributed-ack

gfsh>describe region --name=region2
..........................................................
Name             : region2
Data Policy      : replicate
Hosting Members  : server5
                   server4

Non-Default Attributes Shared By Hosting Members

 Type  |    Name     | Value
------ | ----------- | ---------------
Region | data-policy | REPLICATE
       | size        | 0
       | scope       | distributed-ack
```

This new cluster uses the same configuration as the original system. You can start any number of servers using this cluster configuration. All servers will receive the cluster-level configuration. Servers that specify `group1` also receive the `group1` configuration.

18. Shut down your cluster using the following commands:

```
gfsh>shutdown --include-locators=true
As a lot of data in memory will be lost, including possibly events in queues,
  do you really want to shutdown the entire distributed system? (Y/n): Y
Shutdown is triggered

gfsh>
No longer connected to 192.0.2.0[1099].
```

Deploying Application JARs to VMware GemFire Members

You can dynamically deploy your application JAR files to specific members or to all members in your cluster. VMware GemFire automatically keeps track of JAR file versions; autoloads the deployed JAR files to the CLASSPATH; and auto-registers any functions that the JAR contains.

To deploy and undeploy application JAR files in VMware GemFire, use the `gfsh deploy` or `undeploy` command. You can deploy a single JAR or multiple JARs (by either specifying the JAR filenames or by specifying a directory that contains the JAR files), and you can also target the deployment to a member group or multiple member group. For example, after connecting to the cluster where you want to deploy the JAR files, you could type at the `gfsh` prompt:

```
gfsh> deploy --jars=group1_functions.jar
```

This command deploys the `group1_functions.jar` file to all members in the cluster.

To deploy the JAR file to a subset of members, use the `--groups` argument. For example:

```
gfsh> deploy --jars=group1_functions.jar --groups=MemberGroup1
```

In the example it is assumed that you have already defined the member group that you want to use when starting up your members. See Configuring and Running a Cluster for more information on how to define member groups and add a member to a group.

To deploy all the JAR files that are located in a specific directory to all members:

```
gfsh> deploy --dir=libs/group1-libs
```

You can either provide a JAR file name or a directory of JARs for deployment, but you cannot specify both at once.

To undeploy all previously deployed JAR files throughout the cluster:

```
gfsh> undeploy
```

To undeploy a specific JAR file:

```
gfsh> undeploy --jars=group1_functions.jar
```

To target a specific member group when undeploying all JAR files:

```
gfsh> undeploy --groups=MemberGroup1
```

Only JAR files that have been previously deployed on members in the MemberGroup1 group will be undeployed.

To see a list of all deployed JAR files in your cluster:

```
gfsh> list deployed
```

To see a list of all deployed JARs in a specific member group:

```
gfsh> list deployed --groups=MemberGroup1
```

Sample output:

```

 Member    |     Deployed JAR     |                  JAR Location
--------- | -------------------- | --------------------------------------------------
datanode1 | group1_functions.jar | /usr/local/gemfire/deploy/group1_functions.v1.jar
datanode2 | group1_functions.jar | /usr/local/gemfire/deploy/group1_functions.v1.jar
```

For more information on `gfsh` usage, see gfsh.

Deployment Location for JAR Files

The system location where JAR files are written on each member is determined by the `deploy-working-dir` VMware GemFire property configured for that member. For example, you could have the following configured in the `gemfire.properties` file for your member:

```
#gemfire.properties
deploy-working-dir=/usr/local/gemfire/deploy
```

This deployment location can be local or a shared network resource (such as a mount location) used by multiple members in order to reduce disk space usage. If you use a shared directory, you still need to deploy the JAR file on every member that you want to have access to the application, because deployment updates the CLASSPATH and auto-registers functions.

About Deploying JAR Files and the Cluster Configuration Service

By default, the cluster configuration service distributes deployed JAR files to all locators in the cluster. When you start a new server using `gfsh`, the locator supplies configuration files and deployed jar files to the member and writes them to the server's directory.

See Overview of the Cluster Configuration Service.

Versioning of JAR Files

When you deploy JAR files to a cluster or member group, the JAR file is modified to indicate version information in its name. Each JAR filename contains a version number inserted just before the `.jar` suffix. For example, if you deploy `MyClasses.jar` five times, the filename is displayed as `MyClasses.v5.jar` when you list all deployed jars.

When you deploy a new JAR file, the member receiving the deployment checks whether the JAR file is a duplicate, either because the JAR file has already been deployed on that member or because the JAR file has already been deployed to a shared deployment working directory that other members are also using. If another member has already deployed this JAR file to the shared directory (determined by doing a byte-for-byte compare to the latest version in its directory), the member receiving the latest deployment does not write the file to disk. Instead, the member updates the ClassPathLoader to use the already deployed JAR file. If a newer version of the JAR file is detected on disk and is already in use, the deployment is canceled.

When a member begins using a JAR file, the member obtains a shared lock on the file. If the member receives a newer version by deployment, the member releases the shared lock and tries to delete the existing JAR file in favor of the newer version. If no other member has a shared lock on the existing JAR, the existing, older version JAR is deleted.

Automatic Class Path Loading

When a cache is started, the new cache requests that the latest versions of each JAR file in the current working directory be added to the ClassPathLoader. If a JAR file has already been deployed to the ClassPathLoader, the ClassPathLoader updates its loaded version if a newer version is found; otherwise, there is no change. If detected, older versions of the JAR files are deleted if no other member has a shared lock on them.

Undeploying a JAR file does not automatically unload the classes that were loaded during deployment. You need to restart your members to unload those classes.

When a cache is closed it requests that all currently deployed JAR files be removed from the ClassPathLoader.

If you are using a shared deployment working directory, all members sharing the directory should belong to the same member group. Upon restart, all members that share the same deployment working directory will deploy and autoload their CLASSPATH with any JARs found in the current working directory. This means that some members may load the JARs even though they are not part of the member group that received the original deployment.

Automatic Function Registration

When you deploy a JAR file that contains a function (in other words, contains a class that implements the Function interface), the function is automatically registered through the `FunctionService.registerFunction` method. If another JAR file is deployed (either with the same JAR filename or another filename) with the same function, the new implementation of the function is registered, overwriting the old one. If a JAR file is undeployed, any functions that were auto-registered at the time of deployment are unregistered. Because deploying a JAR file that has the same name multiple times results in the JAR being un-deployed and re-deployed, functions in the JAR are unregistered and re-registered each time this occurs. If a function with the same ID is registered from multiple differently named JAR files, the function is unregistered if any of those JAR files are re-deployed or un-deployed.

During `cache.xml` load, the parameters for any declarables are saved. If functions found in a JAR file are also declarable, and have the same class name as the declarables whose parameters were saved after loading cache.xml, then function instances are created using those Parameters and are also registered. Therefore, if the same function is declared multiple times in the `cache.xml` with different sets of parameters, when the JAR is deployed a function is instantiated for each set of parameters. If any functions are registered using parameters from a `cache.xml` load, the default, no-argument function is not registered.

Using Member Groups

VMware GemFire allows you to organize your cluster members into logical member groups.

The use of member groups in VMware GemFire is optional. The benefit of using member groups is the ability to coordinate certain operations on members based on logical group membership. For example, by defining and using member groups you can:

- Alter a subset of configuration properties for a specific member or members. See alter runtime in `gfsh`.
- Perform certain disk operations like disk-store compaction across a member group. See Disk Store Commands for a list of commands.
- Manage specific indexes or regions across all members of a group.
- Start and stop multi-site (WAN) services such as gateway senders and gateway receivers across a member group.
- Deploy or undeploy JAR applications on all members in a group.
- Execute functions on all members of a specific group.

You define group names in the `groups` property of your member's `gemfire.properties` file or upon member startup in `gfsh`.

Note: Any roles defined in the currently existing `roles` property will now be considered a group. If you wish to add membership roles to your cluster, you should add them as member groups in the `groups` property. The `roles` property has been deprecated in favor of using the `groups` property.

To add a member to a group, add the name of a member group to the `gemfire.properties` file of the member prior to startup or you can start up a member in `gfsh` and pass in the `--groups` argument at startup time.

A single member can belong to more than one group.

Member groups can also be used to organize members from either a client's perspective or from a peer member's perspective. See Organizing Peers into Logical Member Groups and Organizing Servers Into Logical Member Groups for more information. On the client side, you can supply the member group name when configuring a client's connection pool. Use the <pool server-group> element in the client's cache.xml.

Exporting and Importing Cluster Configurations

The cluster configuration service exports and imports configurations created using `gfsh` for an entire VMware GemFire cluster.

The cluster configuration service saves the cluster configuration as you create regions, disk-stores and other objects using `gfsh` commands. You can export this configuration as well as any jar files that contain application files to a ZIP archive and then import this configuration to create a new cluster.

Exporting a Cluster Configuration

Issue the `gfsh export cluster-configuration` command to save the configuration data for your cluster in a ZIP archive. This ZIP file contains subdirectories for cluster-level configurations and a directory for each group specified in the cluster. The contents of these directories are described in Cluster Configuration Files and Troubleshooting.

To export a cluster configuration, run the `gfsh export cluster-configuration` command while connected to a VMware GemFire cluster. For example:

```
export cluster-configuration --zip-file-name=/home/username/configs/myClusterConfig.zi
p
```

See export cluster-configuration.

Note: `gfsh` only saves cluster configuration values for configurations specified using `gfsh`. Configurations created by the management API are not saved with the cluster configurations.

Importing a Cluster Configuration

Use the `gfsh import cluster-configuration` command to configure a new cluster based on a configuration exported from another system. You can import a cluster configuration only into a new cluster, or into a running cluster that has not yet been configured and contains no defined regions. That is, when:

- There are no running cache servers

or

- The only running cache servers meet all of the following criteria:
  - Have been recently started

- Have no regions defined in them
- Have been given no other configuration changes since they started

After you have imported the configuration, any servers you start receive this cluster configuration.

To import a cluster configuration, start one or more locators and then run the `gfsh import cluster-configuration` command. For example:

```
import cluster-configuration --zip-file-name=/home/username/configs/myClusterConfig.zi
p
```

See import cluster-configuration.

Cluster Configuration Files and Troubleshooting

When you use the cluster configuration service in VMware GemFire, you can examine the generated configuration. The gfsh export cluster-configuration command outputs configured properties, the configuration on a per-group basis or for the entire cluster, and the list of deployed JAR files.

If the output is written to either a ZIP file or an XML file, you can import this configuration to a new cluster. See Exporting and Importing Cluster Configurations.

Upon the deploy of a JAR file, the JAR file is added to a created directory called `cluster_config` within the locator's directory of files. Within `cluster_config` will be another directory named for the member group that has the configuration. For configurations that apply to all members of a cluster, the directory is named either `cluster` or the name specified when starting up the locator with the `--cluster-config-dir` option.

Troubleshooting Tips

- When you start a locator using `gfsh`, you should see the following message:

  ```
  Cluster configuration service is up and running.
  ```

  If you do not see this message, there may be a problem with the cluster configuration service. Use the `status cluster-config-service` command to check the status of the cluster configuration.

  - If the command returns RUNNING, the cluster configuration is running normally.
  - If the command returns WAITING, run the `status locator` command. The output of this command returns the cause of the WAITING status.

- When using a `cache.xml` file for configuration, there is a specific order to the application of the configuration in these files. VMware GemFire applies the cluster-wide configuration files first. Group-level configurations follow. Last will be the configuration in a member's own configuration files (`cache.xml` and `gemfire.properties` files).
- If a server start fails with the following exception: `ClusterConfigurationNotAvailableException`, the cluster configuration service may not be in the RUNNING state. Because the server requests the cluster configuration from the locator, which is not available, the `start server` command fails.

- You can determine what configurations a server received from a locator by examining the server's log file. See Logging.
- If a `start server` command specifies a cache.xml file that conflicts with the existing cluster configuration, the server startup may fail.
- If a `gfsh` command fails because the cluster configuration cannot be saved, the following message displays:

  ```
  Failed to persist the configuration changes due to this command,
  Revert the command to maintain consistency. Please use "status cluster-config-s
  ervice"
  to determine whether Cluster configuration service is RUNNING."
  ```

- There are some types of configurations that cannot be made using `gfsh`. See gfsh Limitations.

Sizing a GemFire Cluster

This topic describes GemFire cluster sizing.

Overview

Sizing a GemFire deployment is a process that involves calculation, as well as experimentation and testing. Some experimentation and testing is required to arrive at reasonably accurate values for the key sizing parameters that will work well in practice. This experimentation and testing involves representative data and workload, starting at a very small scale.

Experimentation and testing is required because memory overhead can vary greatly due to variations in data and workload. This makes it impractical to calculate the overhead precisely, as it is a function of too many variables, many of which stem from the Java runtime environment (JVM) and its memory management mechanism.

Resource Considerations

Memory is the primary means of data storage in GemFire and is the first resource to consider for sizing purposes.

Horizontal scaling to satisfy memory requirements also scales out all the other hardware resources, the CPU, network, and disk. Because of this, when memory requirements are satisfied and the adequate cluster size is determined, often only small adjustments are needed to cover all the other required resources and complete the sizing process.

Typically, memory drives horizontal scaling, but it can be any of the hardware resources. In addition to hardware resources, soft resources should be considered. The most important software resources to consider are file descriptors, mostly for sockets in use, and threads (processes).

Experimentation and Testing

To size a GemFire cluster:

1. Deploy a small representative data set and workload in a small cluster.

2. Tune the cluster to the desired performance.
3. Scale out the cluster while ensuring that key performance metrics stay within the desired SLA.

Testing at full scale is ideal, if you have sufficient resources available to use in testing. If sufficient resources are not available to use in testing, you can scale out multiple times, a few nodes at a time, to provide data points to use to project resource usage at full scale. This is an iterative process that involves analysis and tuning at each step. GemFire statistics can assist in this analysis. can be aided greatly by GemFire statistics.

For large scale deployments that involve large data volumes, the general guideline is to scale vertically as much as possible to fit as much data as possible in a single GemFire instance. This helps minimize the size of the cluster. The limit to vertical scaling may depend on the desired SLA around node failure.

Requirements and Assumptions

To maximize the accuracy of your GemFire cluster sizing, and to minimize unexpected situations in your production environment, VMware recommends that you first run tests to characterize memory and other resource usage under a representative workload.

Requirements:

- A subset of representative data. Typically, representative data that more closely matches the real data produces more accurate results.
- A matching subset of workload that matches the production workload as closely as possible.
- Hardware resources for testing, ideally the same category as would be used in production: the same CPU, memory, network, and disk resources per node. At a minimum, you must be able to run three GemFire data nodes to start, then be able to add a few more node to validate the scalability. In addition, you must have the same number of hosts for GemFire clients to be able to create an adequate workload.
- Familiarity with key GemFire concepts and features, such as partitioned regions, serialization, etc.

You should follow the documented best practices, such as the JVM GC configuration (CMS and ParNew), and use the currently supported platforms.

Architectural and Design Considerations

Before a sizing effort can start, the overarching architectural decisions have to be made, such as which GemFire regions to use for different types of data or what redundancy level to use. The results of sizing can inform architectural and design decisions for which multiple options are possible.

Serialization

Serialization can make a significant difference in the per-entry data overhead in memory, and subsequently in the overall memory requirements.

GemFire's PDX serialization is a serialization format that keeps data in a usable serialized form. This allows most operations on data entries without having to deserialize them, resulting in both space and performance benefits. These qualities make the PDX serialization the recommended serialization approach for most use cases.

DataSerializable is another GemFire serialization mechanism that is more space efficient than either PDX or Java Serializable. However, unlike PDX, it requires deserialization on any kind of access.

Per-entry Memory Overhead

Listed below are factors that can have significant impact on the memory overhead for data on a per entry basis, as well as performance:

- **Choice of GemFire region type**: Different regions have different per-entry overheads. This overhead is documented below and is included in the sizing spreadsheet.
- **Choice of the serialization mechanism**: GemFire offers multiple serialization options, as well as the ability to have values stored serialized. As mentioned above, GemFire PDX serialization is the generally recommended serialization mechanism due to its space and performance benefits.
- **Choice of Keys**: Smaller and simpler keys are more efficient in terms of both space and performance.
- **Use of indexes**: Indexing incurs a per-entry overhead, as documented in Memory Requirements for Cached Data.

For more detailed information and guidelines, see Memory Requirements for Cached Data.

If the data value objects are small but great in number, the per-entry overhead can add up to a significant memory requirement. You can reduce this overhead by grouping multiple data values into a single entry or by using containment relationships. For example, you can choose to have your Order objects contain their line items instead of having a separate OrderLineItems region. If this option is available, using it may yield performance improvements in addition to space savings.

Partitioned Region Scalability

GemFire partitioned regions scale out by rebalancing their data buckets (partitions) to distribute the data evenly across all available nodes in a cluster. When new nodes are added to the cluster, rebalancing causes some buckets to move from the old to the new nodes such that the data is evenly balanced across all the nodes. For this to work effectively, with the end result being a well-balanced cluster, there should be at least one order of magnitude more buckets than data nodes for each partitioned region.

Typically, increasing the number of buckets improves data distribution. However, since the number of buckets cannot be changed dynamically and without downtime, the projected horizontal scale-out taken into account when determining the optimal number of buckets. Otherwise, as the system scales out over time, the data may become less evenly distributed. In the extreme case, when the number of nodes exceeds the number of buckets, adding new nodes has no effect, and the ability to scale out is lost.

Related to this is the choice of data partitioning scheme, the goal of which is to yield even data and workload distribution in the cluster. If problem with the partitioning scheme exists, the data, and likely the workload, will not be evenly balanced, and scalability will be lost.

Redundancy

Typically, choice of redundancy may be driven by data size and by whether or not data can be retrieved from some other backing store besides GemFire. Other considerations might also be a factor in the decision.

For example, GemFire can be deployed in an active/active configuration in two data centers such that each can take on the entire load, but will do so only if necessitated by a failure. Typically, in such deployments there are four live copies of the data at any time, with two in each datacenter. If two nodes failed in a single datacenter, the other datacenter would take over the entire workload until those two nodes were restored. This avoids data loss in the first datacenter. You could instead set redundancy to two, for a total of three copies of data). This would provide high availability even in case of a single node failure, and avoids paying the price of rebalancing when a single node fails. In this case, instead of rebalancing, a single failed node is restarted, while two copies of data still exist.

Relationship Between Horizontal and Vertical Scale

For deployments that can grow very large, you should allow for the growth by taking advantage of not just horizontal scalability, but also the ability to store as much data as possible in a single node. GemFire has been deployed in clusters of over 100 nodes. However, smaller clusters are easier to manage. So, as a general rule, you should store as much data as possible in a single node while maintaining a comfortable data movement requirement for re-establishing the redundancy SLA after a single point of failure. GemFire has been used with heaps of well over 64GB in size..

NUMA Considerations

You should understand the Non-Uniform Memory Architecture (NUMA) memory boundaries when deciding on the JVM size, and VM size in virtualized deployments.

Most modern CPUs implement this kind of architecture where memory is divided across the CPUs such that memory directly connected to the bus of each CPU has very fast access whereas memory accesses by that same CPU on the other portions of memory (directly connected to the other CPUs) can pay a significant wait-state penalty for accessing data. An example is a system that has four CPUs with eight cores each and a Non-Uniform Memory Architecture that assigns each CPU its own portion of the memory. As an example, assume that the total memory on the machine is 256GB. In this case, each NUMA node is 64GB. Growing a JVM larger than 64GB on this machine will cause wait-states to be induced when the CPUs must cross NUMA node boundaries to access memory within the heap. For this reason, you should size GemFire JVMs to fit within a single NUMA node to optimize performance.

GemFire Queues

If any GemFire queueing capabilities are used, such as for WAN distribution, client subscription, or asynchronous event listeners, you should evaluate the queues' capacity in the context of the desired SLA. For example, for how long should gateway or client subscription queues be able to keep queueing events when the connection is lost? Given that, how large should the queues be able to grow? An effective way to learn the answers to these kinds of questions is to watch the queues' growth during sizing experiments, using GemFire statistics. For more information about this, see Step 3: Vertical Sizing below.

For WAN distribution, you should evaluate the distribution volume requirements and ensure adequate network bandwidth sizing. If sites connected by the WAN gateway may be down for extended periods of time, such as days or weeks, you must overflow the gateway queues to disk and ensure that you have sufficient disk space for those queues. If you have insufficient disk space for the queues, you may need to shut off the Gateway senders to prevent running out of space.

Sizing Process

To size a GemFire cluster:

1. Domain object sizing: Produce an entry size estimate for all the domain objects. Use this with number of entries to estimate the total memory requirements.
2. Estimating total memory and system requirements: Based on the data sizes, estimate the total memory and system requirements using the sizing spreadsheet, which takes into account GemFire region overhead. This does not account for other overhead, but provides a starting point.
3. Vertical sizing: Use the results of the previous step as the starting point in configuring a three-node cluster. Vertical sizing determines the “building block” – the sizing, configuration, and workload for a single node – by experimentation.
4. Scale-out validation: Iteratively test and adjust the single “building block” node from the previous step to verify near-linear scalability and performance.
5. Projection to full scale: Use the results of scale-out validation to arrive at the sizing configuration that meets your desired capacity and SLA.

The following sections provide details about each step.

Step 1: Domain object sizing

Before you can make any other estimates, you must estimate the size of the domain objects to be stored in the cluster.

An effective way to size a domain object is to run a single instance GemFire test with GemFire statistics enabled. In this instance, store each domain object to be sized in a dedicated partitioned region. The test loads a number of instances of each domain object, making sure they all stay in memory, with no overflow. After running the test, load the statistics file from it into VSD and examine `dataStoreBytesInUse` and `dataStoreEntryCount` partition region stats for each partitioned region. Dividing the value of `dataStoreBytesInUse` by the value of `dataStoreEntryCount` provides an estimate for the average value size that is as accurate as is possible.

Note: This estimate does not include the key size and entry overhead.

Another way to size domain objects is to use a heap histogram. In this approach you should run a separate test for each domain object. This simplifies the process of determining what classes are associated with data entries, based on the number of entries in memory, to figure out what classes

Step 2: Estimating total memory and system requirements

You can use the System Sizing Worksheet to approximate your total memory and system requirements. The *System Sizing Worksheet* takes into account all the GemFire region related per-entry overhead, and the desired memory headroom.

The spreadsheet formulas are rough approximations that serve to inform a high-level estimate, as they do not account for any other overhead such as buffers, threads, queues, application workload, etc. Additionally, the results obtained from the spreadsheet do not have any performance context. For this reason, the steps in Step 3: Vertical Sizing use the results for memory allocation per server obtained from the spreadsheet as the starting point for the vertical sizing process.

Step 3: Vertical Sizing

Use vertical sizing to determine what fraction of the total requirements for storage and workload can be satisfied with a single data node, and with what resources. This represents a “building block” (a unit of scale) and includes both the size of the resources and the workload capacity. It also includes the complete configuration of the building block (system, VM if present, JVM, and GemFire).

For example, a result of this step for a simple read-only application might be that a single data node with a JVM sized to 64G can store 40G of data and support a workload of 5000 get operations per second within the required latency SLA, without exhausting any resources. You should capture all the key performance indicators for the application, and verify that they meet the desired SLA. A complete output of the vertical sizing step includes all the relevant details such as hardware resources per node, peak capacity, and performance at peak capacity, and notes which resource becomes a bottleneck at peak capacity.

This approach uses experimentation to determine the optimal values for all relevant configuration settings, including the system, VM if virtualization is used, JVM, and GemFire configuration to be used.

To run experiments and tests, you musy have a cluster of three data nodes and a locator, as well as additional hosts to run clients to generate the application workload. Three data nodes are required to fully exercise the partitioning of data in partitioned regions across multiple nodes in presence of data redundancy. As a starting point, the data nodes should be sized based on the estimates obtained from the sizing spreadsheet completed in Step 2: Estimating total memory and system requirements.

Typically, the following configuration is used to begin:

- A heap headroom of 50% of the old generation
- `CMSInitiatingOccupancyFraction` is set to 65%
- The young generation is sized to 10% of the total heap

GemFire logging and statistics should be enabled for all the test runs. The logs should be routinely checked for problems. The statistics are analyzed for problems, verification of resources, and performance. Performance metrics can be collected by the application test logic as well. Any relevant latency metrics must be collected by the test application.

If WAN distribution is needed, you should set up an identical twin cluster and configure the WAN distribution between the two clusters. You should also size WAN capacity.

Test runs should exercise a representative application workload, with duration long enough to incur multiple GC cycles, so that stable resource usage can be confirmed. If any GemFire queues are used, run tests to determine adequate queue sizes that meet the SLA. If disk storage is used, determine adequate disk store size and configuration per node as part of this exercise.

After each test run, examine the latency metrics collected by the application. Use VSD to examine the statistics and correlate the resource usage with latencies and throughput observed. You should examine the following:

- Memory (heap, and non-heap, GC)
- CPU
- System load
- Network
- File descriptors
- Threads
- Queue statistics

For information about VSD and these statistics, see Quick Guide to Useful Statistics.

One of the objectives of vertical sizing is to determine the headroom required to accomplish the desired performance. This might take several tests to tune the headroom to no more and no less than needed. A much larger headroom than needed could amount to a significant waste of resources. A smaller headroom could cause higher GC activity and CPU usage and hurt performance.

Locator Sizing

Locator JVM sizing may be necessary when JMX Manager is running in the locator JVM, and JMX is used for monitoring. An effective way to do this is to set the locator heap to 0.5G and monitor it during the scale-out.

Notes on GC

The most important goal of GC is to avoid full GCs, as they cause pauses which can result in a GemFire data node to be unresponsive, and, as a result, be expelled from the cluster. The permanent generation space can trigger a full GC as well, which happens when it fills completely. You should size this appropriately to avoid this. Additionally, you can instruct the JVM to garbage collect the permanent generation space along with CMS GC using the following option:

```
-XX:+CMSClassUnloadingEnabled
```

You can tune GC for two of the following three:

- Latency
- Throughput
- Memory footprint

Heap headroom is important because with GemFire we sacrifice the memory footprint to accomplish latency and throughput goals.

Long minor GC pauses can be shortened by reducing the young generation. This will likely increase the frequency of minor collections. Additionally, for very large heaps, for example those of 64G and above, the old generation impact on minor GC pauses may be reduced by using the following GC settings:

```
-XX:+UnlockDiagnosticVMOptions XX:ParGCCardsPerStrideChunk=32768
```

Step 4: Scale-out Validation

During this step, you scale out the initial three node cluster at least twice, adding at least a few nodes each time. You should also scale out the client hosts accordingly to be able to create adequate workload at each step. You should increase the workload proportionally to the scale-out.

There is no definitive rule about how much to increase the cluster size, or in what increments. Typically, this determination is dictated by available hardware resources.

The goal of this step is to validate the “building block” configuration and capacity at some, larger than initial, scale. This allows you to project the capacity to full scale with confidence. You may need to tune the configuration at various points. For example, when you add more nodes to the cluster, more socket connections, buffers, and threads will be in use on each node, resulting in higher memory usage per node (both heap and non-heap), as well as increased file descriptors in use.

If you use JMX for monitoring, watch the heap usage of the locator running the JMX Manager.

Step 5: Projection to Full Scale

After you have completed Step 4: Scale-out Validation, you can determine the total cluster size. You know the storage and workload capacity of a single node and that you can scale horizontally to meet the full requirements. Additionally, you have already tuned the cluster configuration to meet the demands of the application workload.

Sizing Quick Reference

General recommendations to use as a starting point in capacity planning and sizing

Data Node Heap Size	**Use**
Up to 32GB	Smaller data volumes (up to a few hundred GB); very low latency required
64GB+	Larger data volumes (500GB+)
CPU Cores per Data Node	**Use**
2 to 4	Development; smaller heaps
6 to 8	Production; performance/system testing; larger heaps
Network Bandwidth	**Use**
1GbE	Development
High bandwidth (e.g. 10GbE)	Production; performance/system testing
Disk Storage	**Use**

Data Node Heap Size	Use
DAS, or SAN	Always
NAS	Do not use; performance and resilience issues

- **Memory/CPU relationship**: mind the NUMA boundary
- **Virtualization**: Do not oversubscribe resources (memory, CPU, storage). Run a single GemFire data node JVM per VM.

Using gfsh to Manage a Remote Cluster Over HTTP or HTTPS

You can connect `gfsh` via HTTP or HTTPS to a remote cluster and manage the cluster using `gfsh` commands.

To connect `gfsh` using the HTTP protocol to a remote cluster:

1. Launch `gfsh`. See Starting gfsh.
2. When starting the remote cluster on the remote host, you can optionally specify `--http-bind-address` and `--http-service-port` as VMware GemFire properties when starting up your JMX manager (server or locator). These properties can be then used in the URL used when connecting from your local system to the HTTP service in the remote cluster. For example:

   ```
   gfsh>start server --name=server1 --J=-Dgemfire.jmx-manager=true \
   --J=-Dgemfire.jmx-manager-start=true --http-service-port=8080 \
   --http-service-bind-address=myremotecluster.example.com
   ```

   This command must be executed directly on the host machine that will ultimately act as the remote server that hosts the HTTP service for remote administration. (You cannot launch a server remotely.)
3. On your local system, run the `gfsh connect` command to connect to the remote system. Include the `--use-http` and `--url` parameters. For example:

   ```
   gfsh>connect --use-http=true --url="http://myremotecluster.example.com:8080/geo
   de/v1"

   Successfully connected to: VMware GemFire Manager's HTTP service @ http://myrem
   otecluster.example.com:8080/geode/v1
   ```

   See connect.

   `gfsh` is now connected to the remote system. Most `gfsh` commands will now execute on the remote system; however, there are exceptions. The following commands are executed on the local cluster: - `alter disk-store` - `compact offline-disk-store` - `describe offline-disk-store` - `help` - `hint` - `sh` (for executing OS commands) - `sleep` - `start jconsole` (however, you can connect JConsole to a remote cluster when gfsh is connected to the cluster via JMX) - `start jvisualvm` - `start locator` - `start server` - `start vsd` - `status locator``*` - `status server``*` - `stop locator``*` - `stop server``*` - `run` (for executing gfsh scripts) - `validate disk-store` - `version`

> *You can stop and obtain the status of *remote locators and servers* when `gfsh` is connected to the cluster via JMX or HTTP/S by using the `--name` option for these `stop`/`status` commands. If you are using the `--pid` or `--dir` option for these commands, then the`stop`/`status` commands are executed only locally.

To configure SSL for the remote connection (HTTPS), enable SSL for the `http` component in gemfire.properties or gfsecurity-properties or upon server startup. See SSL for details on configuring SSL parameters. These SSL parameters also apply to all HTTP services hosted on the configured JMX Manager, which can include the following:

- Developer REST API service
- Pulse monitoring tool

Deploying Configuration Files without the Cluster Configuration Service

You can deploy your VMware GemFire configuration files in your system directory structure or in jar files. You determine how you want to deploy your configuration files and set them up accordingly.

Note: If you use the cluster configuration service to create and manage your VMware GemFire cluster configuration, the procedures described in this section are not needed because VMware GemFire automatically manages the distribution of the configuration files and jar files to members of the cluster. See Overview of the Cluster Configuration Service.

You can use the procedures described in this section to distribute configurations that are member-specific, or for situations where you do not want to use the cluster configuration service.

- **Main Steps to Deploying Configuration Files**

  These are the basic steps for deploying configuration files, with related detail in sections that follow.

- **Default File Specifications and Search Locations**

  Each file has a default name, a set of file search locations, and a system property you can use to override the defaults.

- **Changing the File Specifications**

  You can change all file specifications in the `gemfire.properties` file and at the command line.

- **Deploying Configuration Files in JAR Files**

  This section provides a procedure and an example for deploying configuration files in JAR files.

Main Steps to Deploying Configuration Files

These are the basic steps for deploying configuration files, with related detail in sections that follow.

1. Determine which configuration files you need for your installation.
2. Place the files in your directories or jar files.

3. For any file with a non-default name or location, provide the file specification in the system properties file and/or in the member `CLASSPATH`.

VMware GemFire Configuration Files

- `gemfire.properties`. Contains the settings required by members of a cluster. These settings include licensing, system member discovery, communication parameters, logging, and statistics. See the VMware GemFire Properties Reference.
- `gfsecurity.properties`. An optional separate file that contains security-related (`security-*`) settings that are otherwise defined in `gemfire.properties`. Placing these member properties into a separate file allows you to restrict user access to those specific settings. See the VMware GemFire Properties Reference.
- `cache.xml`. Declarative cache configuration file. This file contains XML declarations for cache, region, and region entry configuration. You also use it to configure disk stores, database login credentials, server and remote site location information, and socket information. See cache.xml.

Default File Specifications and Search Locations

Each file has a default name, a set of file search locations, and a system property you can use to override the defaults.

To use the default specifications, place the file at the top level of its directory or jar file. The system properties are standard file specifications that can have absolute or relative pathnames and filenames.

Note: If you do not specify an absolute file path and name, the search examines all search locations for the file.

Default File Specification	Search Locations for Relative File Specifications	Available Property for File Specification
`gemfire.properties`	1. current directory 2. home directory 3. CLASSPATH	As a Java system property, use `gemfirePropertyFile`
`cache.xml`	1. current directory 2. CLASSPATH	In `gemfire.properties`, use the `cache-xml-file` property

Examples of valid `gemfirePropertyFile` specifications:

- `/zippy/users/jpearson/gemfiretest/gemfire.properties`
- `c:\gemfiretest\gemfire.prp`
- `myGF.properties`
- `test1/gfprops`

For the `test1/gfprops` specification, if you launch your VMware GemFire system member from `/testDir` in a Unix file system, VMware GemFire looks for the file in this order until it finds the file or exhausts all locations:

1. `/testDir/test1/gfprops`
2. `<yourHomeDir>/test1/gfprops`
3. under every location in your `CLASSPATH` for `test1/gfprops`

Changing the File Specifications

You can change all file specifications in the `gemfire.properties` file and at the command line.

Note: VMware GemFire applications can use the API to pass `java.lang.System` properties to the cluster connection. This changes file specifications made at the command line and in the `gemfire.properties` file. You can verify an application's property settings in the configuration information logged at application startup. The configuration is listed when the `gemfire.properties` `log-level` is set to `config` or lower.

This invocation of the application, `testApplication.TestApp1`, provides non-default specifications for both the `cache.xml` and `gemfire.properties` files:

```
java -Dgemfire.cache-xml-file=\
/gemfireSamples/examples/dist/cacheRunner/queryPortfolios.xml \
-DgemfirePropertyFile=defaultConfigs/gemfire.properties \
testApplication.TestApp1
```

The gfsh start server command can use the same specifications:

```
gfsh>start server \
--J=-Dgemfire.cache-xml-file=/gemfireSamples/examples/dist/cacheRunner/queryPortfolio
s.xml \
--J=-DgemfirePropertyFile=defaultConfigs/gemfire.properties
```

You can also change the specifications for the `cache.xml` file inside the `gemfire.properties` file.

Note: Specifications in `gemfire.properties` files cannot use environment variables.

Example `gemfire.properties` file with non-default `cache.xml` specification:

```
#Tue May 09 17:53:54 PDT 2006
mcast-address=192.0.2.0
mcast-port=10333
locators=cache-xml-file=/gemfireSamples/examples/dist/cacheRunner/queryPortfolios.xml
```

Deploying Configuration Files in JAR Files

This section provides a procedure and an example for deploying configuration files in JAR files.

Procedure

1. Jar the files.
2. Set the VMware GemFire system properties to point to the files as they reside in the jar file.
3. Include the jar file in your `CLASSPATH`.

4. Verify the jar file copies are the only ones visible to the application at runtime. VMware GemFire searches the `CLASSPATH` after searching other locations, so the files cannot be available in the other search areas.
5. Start your application. The configuration file is loaded from the jar file.

Example of Deploying a Configuration JAR

The following example deploys the cache configuration file, `myCache.xml`, in `my.jar`. The following displays the contents of `my.jar`:

```
% jar -tf my.jar
META-INF
META-INF/MANIFEST.MF
myConfig/
myConfig/myCache.xml
```

In this example, you would perform the following steps to deploy the configuration jar file:

1. Set the system property `gemfire.cache-xml-file` to `myConfig/myCache.xml`.
2. Set your `CLASSPATH` to include `my.jar`.
3. Verify there is no file already in the filesystem named `./myConfig/myCache.xml`, so VMware GemFire will be forced to search the jar file to find it.

When you start your application, the configuration file is loaded from the jar file.

Starting Up and Shutting Down Your System

Determine the proper startup and shutdown procedures, and write your startup and shutdown scripts.

Well-designed procedures for starting and stopping your system can speed startup and protect your data. The processes you need to start and stop include server and locator processes and your other VMware GemFire applications, including clients. The procedures you use depend in part on your system's configuration and the dependencies between your system processes.

Use the following guidelines to create startup and shutdown procedures and scripts. Some of these instructions use `gfsh`.

Starting Up Your System

You should follow certain order guidelines when starting your VMware GemFire system.

Start servers before you start their client applications. In each cluster, follow these guidelines for member startup:

- Start locators first. See Running VMware GemFire Locator Processes for examples of locator start up commands.
- Start cache servers before the rest of your processes unless the implementation requires that other processes be started ahead of them. See Running VMware GemFire Server Processes for examples of server start up commands.

- If your cluster uses both persistent replicated and non-persistent replicated regions, you should start up all the persistent replicated members in parallel before starting the non-persistent regions. This way, persistent members will not delay their startup for other persistent members with later data.
- For a system that includes persistent regions, see Start Up and Shut Down with Disk Stores.
- If you are running producer processes and consumer or event listener processes, start the consumers first. This ensures the consumers and listeners do not miss any notifications or updates.
- If you are starting up your locators and peer members all at once, you can use the `locator-wait-time` property (in seconds) upon process start up. This timeout allows peers to wait for the locators to finish starting up before attempting to join the cluster.

  If the process cannot initially reach a locator, it will sleep for `join-retry-sleep` milliseconds between retries until it either connects or the number of seconds specified in `locator-wait-time` has elapsed. By default, `locator-wait-time` is set to zero meaning that a process that cannot connect to a locator upon startup will throw an exception.

Note: You can optionally override the default timeout period for shutting down individual processes. This override setting must be specified during member startup. See Shutting Down the System for details.

Starting Up After Losing Data on Disk

This information pertains to catastrophic loss of VMware GemFire disk store files. If you lose disk store files, your next startup may hang, waiting for the lost disk stores to come back online. If your system hangs at startup, use the `gfsh` command `show missing-disk-store` to list missing disk stores and, if needed, revoke missing disk stores so your system startup can complete. You must use the Disk Store ID to revoke a disk store. These are the two commands:

```
gfsh>show missing-disk-stores

Disk Store ID                      |   Host    |               Directory
------------------------------------ | --------- | -------------------------------------
--
60399215-532b-406f-b81f-9b5bd8d1b55a | excalibur | /usr/local/gemfire/deploy/disk_stor
e1

gfsh>revoke missing-disk-store --id=60399215-532b-406f-b81f-9b5bd8d1b55a
```

Note: This `gfsh` commands require that you are connected to the cluster via a JMX Manager node.

Shutting Down the System

Shut down your VMware GemFire system by using either the `gfsh shutdown` command or by shutting down individual members one at a time.

Using the shutdown Command

If you are using persistent regions, (members are persisting data to disk), you should use the `gfsh shutdown` command to stop the running system in an orderly fashion. This command synchronizes persistent partitioned regions before shutting down, which makes the next startup of the cluster as efficient as possible.

If possible, all members should be running before you shut them down so synchronization can occur. Shut down the system using the following `gfsh` command:

```
gfsh>shutdown
```

By default, the shutdown command will only shut down data nodes. If you want to shut down all nodes including locators, specify the `--include-locators=true` parameter. For example:

```
gfsh>shutdown --include-locators=true
```

This will shut down all locators one by one, shutting down the manager last.

To shutdown all data members after a grace period, specify a time-out option (in seconds).

```
gfsh>shutdown --time-out=60
```

To shutdown all members including locators after a grace period, specify a time-out option (in seconds).

```
gfsh>shutdown --include-locators=true --time-out=60
```

Shutting Down System Members Individually

If you are not using persistent regions, you can shut down the cluster by shutting down each member in the reverse order of their startup. (See Starting Up Your System for the recommended order of member startup.)

Shut down the cluster members according to the type of member. For example, use the following mechanisms to shut down members:

- Use the appropriate mechanism to shut down any VMware GemFire-connected client applications that are running in the cluster.
- Shut down any cache servers. To shut down a server, issue the following `gfsh` command:

  ```
  gfsh>stop server --name=<...>
  ```

  or

  ```
  gfsh>stop server --dir=<server_working_dir>
  ```

- Shut down any locators. To shut down a locator, issue the following `gfsh` command:

  ```
  gfsh>stop locator --name=<...>
  ```

  or

```
gfsh>stop locator --dir=<locator_working_dir>
```

- Do not use the command line `kill -9` to shut down a server under ordinary circumstances. Especially on systems with a small number of members, using a `kill` instead of a `gfsh stop` can cause the partition detection mechanism to place the system in an end state that will wait forever to reconnect to the terminated server, and there will be no way to restart that terminated server. If a `kill` command appears the only way to rid the system of a server, then `kill` *all* the processes of the cluster or use `kill -INT`, which will allow an orderly shutdown of the process.

Option for System Member Shutdown Behavior

The `DISCONNECT_WAIT` command line argument sets the maximum time for each individual step in the shutdown process. If any step takes longer than the specified amount, it is forced to end. Each operation is given this grace period, so the total length of time the cache member takes to shut down depends on the number of operations and the `DISCONNECT_WAIT` setting. During the shutdown process, VMware GemFire produces messages such as:

```
Disconnect listener still running
```

The `DISCONNECT_WAIT` default is 10000 milliseconds.

To change it, set this system property on the Java command line used for member startup. For example:

```
gfsh>start server --J=-DDistributionManager.DISCONNECT_WAIT=<milliseconds>
```

Each process can have different `DISCONNECT_WAIT` settings.

Running VMware GemFire Locator Processes

The locator is a VMware GemFire process that tells new, connecting members where running members are located and provides load balancing for server use.

You can run locators as peer locators, server locators, or both:

- Peer locators give joining members connection information to members already running in the locator’s cluster.
- Server locators give clients connection information to servers running in the locator’s cluster. Server locators also monitor server load and send clients to the least-loaded servers.

By default, locators run as peer and server locators.

You can run the locator standalone or embedded within another VMware GemFire process. Running your locators standalone provides the highest reliability and availability of the locator service as a whole.

Locator Configuration and Log Files

Locator configuration and log files have the following properties:

- When you start a standalone locator using `gfsh`, `gfsh` will automatically load the required JAR file `lib/geode-dependencies.jar` into the CLASSPATH of the JVM process. If you start a standalone locator using the `LocatorLauncher` API, you must specify this JAR file inside the command used to launch the locator process. For more information on CLASSPATH settings in VMware GemFire, see Setting Up the CLASSPATH. You can modify the CLASSPATH by specifying the `--classpath` parameter.
- Locators are members of the cluster just like any other member. In terms of `mcast-port` and `locators` configuration, a locator should be configured in the same manner as a server. Therefore, if there are two other locators in the cluster, each locator should reference the other locators (just like a server member would). For example:

```
gfsh> start locator --name=locator1 --port=9009 --mcast-port=0 \
--locators='host1[9001],host2[9003]'
```

- You can configure locators within the `gemfire.properties` file or by specifying start-up parameters on the command line. If you are specifying the locator's configuration in a properties file, locators require the same `gemfire.properties` settings as other members of the cluster and the same `gfsecurity.properties` settings if you are using a separate, restricted access security settings file.

  For example, to configure both locators and a multicast port in `gemfire.properties:`

```
locators=host1[9001],host2[9003]
mcast-port=0
```

- There is no cache configuration specific to locators.
- For logging output, the locator creates a log file in its current working directory. Log file output defaults to `locator_name.log` in the locator's working directory. If you restart a locator with a previously used locator name, the existing *locator_name*.log file is automatically renamed for you (for example, `locator1-01-01.log` or `locator1-02-01.log`). You can modify the level of logging details in this file by specifying a level in the `--log-level` argument when starting up the locator.
- By default, a locator will start in a subdirectory (named after the locator) under the directory where `gfsh` is executed. This subdirectory is considered the current working directory. You can also specify a different working directory when starting the locator in `gfsh`.
- By default, a locator that has been shutdown and disconnected due to a network partition event or member unresponsiveness will restart itself and automatically try to reconnect to the existing cluster. When a locator is in the reconnecting state, it provides no discovery services for the cluster. See Handling Forced Cache Disconnection Using Autoreconnect for more details.

Locators and the Cluster Configuration Service

Locators use the cluster configuration service to save configurations that apply to all cluster members, or to members of a specified group. The configurations are saved in the Locator's directory and are propagated to all locators in a cluster. When you start servers using `gfsh`, the servers receive the group-level and cluster-level configurations from the locators.

See Overview of the Cluster Configuration Service.

Start the Locator

Use the following guidelines to start the locator:

- **Standalone locator**. Start a standalone locator in one of these ways:
  - Use the `gfsh` command-line utility. See `gfsh` for more information on using `gfsh`. For example:

    ```
    gfsh>start locator --name=locator1

    gfsh> start locator --name=locator2 --bind-address=192.0.2.0 --port=13489
    ```

  - Start the locator using the `main` method in the `org.apache.geode.distributed.LocatorLauncher` class and the Java executable. Specifically, you use the `LocatorLauncher` class API to run an embedded Locator service in Java application processes that you have created. The directory where you execute the java command becomes the working directory for the locator process.
  - When starting up multiple locators, do not start them up in parallel (in other words, simultaneously). As a best practice, you should wait approximately 30 seconds for the first locator to complete startup before starting any other locators. To check the successful startup of a locator, check for locator log files. To view the uptime of a running locator, you can use the `gfsh status locator` command.
- **Embedded (colocated) locator**. Manage a colocated locator at member startup or through the APIs:
  - Use the `gemfire.properties` `start-locator` setting to start the locator automatically inside your VMware GemFire member. See the Reference. The locator stops automatically when the member exits. The property has the following syntax:

    ```
    #gemfire.properties
    start-locator=[address]port[,server={true|false},peer={true|false}]
    ```

    Example:

    ```
    #gemfire.properties
    start-locator=13489
    ```

  - Use `org.apache.geode.distributed.LocatorLauncher` API to start the locator inside your code. Use the `LocatorLauncher.Builder` class to construct an instance of the `LocatorLauncher`, and then use the `start()` method to start a Locator service embedded in your Java application process. The other methods in the `LocatorLauncher` class provide status information about the locator and allow you to stop the locator.

    ```
    import org.apache.geode.distributed.LocatorLauncher;

     public class MyEmbeddedLocator {
    ```

```
    public static void main(String[] args){
        LocatorLauncher locatorLauncher  = new LocatorLauncher.Builder()
          .setMemberName("locator1")
          .setPort(13489)
          .build();

          locatorLauncher.start();

          System.out.println("Locator successfully started");
        }
    }
```

Here's another example that embeds the locator within an application, starts it and then checks the status of the locator before allowing other members to access it:

```
package example;

import ...

class MyApplication implements Runnable {

  private final LocatorLauncher locatorLauncher;

  public MyApplication(final String... args) {
    validateArgs(args);

    locatorLauncher = new LocatorLauncher.Builder()
      .setMemberName(args[0])
      .setPort(Integer.parseInt(args[1])
      .setRedirectOutput(true)
      .build();
  }

  protected void args(final String[] args) {
    ...
  }

  public void run() {
    ...

    // start the Locator in-process
    locatorLauncher.start();

    // wait for Locator to start and be ready to accept member (client) c
onnections
    locatorLauncher.waitOnStatusResponse(30, 5, TimeUnit.SECONDS);

    ...
  }

  public static void main(final String... args) {
    new MyApplication(args).run();
  }

}
```

Then to execute the application, you would run:

```
/working/directory/of/MyApplication$ java \
 -server -classpath "path/to/installation/lib/geode-dependencies.jar:/pat
h/to/application/classes.jar" \
 example.MyApplication Locator1 11235
```

The directory where you execute the java command becomes the working directory for the locator process.

Check Locator Status

If you are connected to the cluster with `gfsh`, you can check the status of a running locator by providing the locator name. For example:

```
gfsh>status locator --name=locator1
```

If you are not connected to a cluster, you can check the status of a local locator by providing the process ID, the Locator's host name and port, or the locator's current working directory. For example:

```
gfsh>status locator --pid=2986
```

or

```
gfsh>status locator --host=host1 --port=1035
```

or

```
$ gfsh status locator --dir=<locator_working_directory>
```

where <*locator_working_directory*> corresponds to the local working directory where the locator is running.

If successful, the command returns the following information (with the JVM arguments that were provided at startup):

```
$ gfsh status locator --dir=locator1
Locator in /home/user/locator1 on ubuntu.local[10334] as locator1 is currently online.
Process ID: 2359
Uptime: 17 minutes 3 seconds
GemFire Version: 8.0.0
Java Version: 1..0_272
Log File: /home/user/locator1/locator1.log
JVM Arguments: -Dgemfire.enable-cluster-configuration=true -Dgemfire.load-cluster-conf
iguration-from-dir=false
 -Dgemfire.launcher.registerSignalHandlers=true -Djava.awt.headless=true -Dsun.rmi.dg
c.server.gcInterval=9223372036854775806
Class-Path: /Users/username/apache_geode/lib/geode-dependencies.jar

Cluster configuration service is up and running.
```

Stop the Locator

If you are connected to the cluster with `gfsh`, you can stop a running locator by providing the locator name. For example:

```
gfsh>stop locator --name=locator1
```

If you are not connected to a cluster, you can stop a local locator by specifying the locator's process ID or the locator's current working directory. For example:

```
gfsh>stop locator --pid=2986
```

or

```
gfsh>stop locator --dir=<locator_working_directory>
```

where <*locator_working_directory*> corresponds to the local working directory where the locator is running.

Locators and Multi-Site (WAN) Deployments

If you use a multi-site (WAN) configuration, you can connect a locator to a remote site when starting the locator.

To connect a new locator process to a remote locator in a WAN configuration, specify the following at startup:

```
gfsh> start locator --name=locator1 --port=9009 --mcast-port=0 \
--J='-Dgemfire.remote-locators=192.0.2.0[9009],198.51.100.0[9009]'
```

Running VMware GemFire Server Processes

A VMware GemFire server is a process that runs as a long-lived, configurable member of a client/server system.

The VMware GemFire server is used primarily for hosting long-lived data regions and for running standard VMware GemFire processes such as the server in a client/server configuration. You can start and stop servers using the following methods:

- The `gfsh` command-line tool.
- Programmatically, through the `org.apache.geode.distributed.ServerLauncher` API. The `ServerLauncher` API can only be used for VMware GemFire Servers that were started with `gfsh` or with the `ServerLauncher` class itself.

Default Server Configuration and Log Files

The `gfsh` utility uses a working directory for its configuration files and log files. These are the defaults and configuration options:

- When you start a standalone server using `gfsh`, `gfsh` will automatically load the required JAR file `lib/geode-dependencies.jar` into the CLASSPATH of the JVM process. If you start a standalone server using the ServerLauncher API, you must specify this JAR file inside your command to launch the process. For more information on CLASSPATH settings in VMware GemFire, see Setting Up the CLASSPATH.

- Servers are configured like any other VMware GemFire process, with `gemfire.properties` and shared cluster configuration files. It is not programmable except through application plug-ins. Typically, you provide the `gemfire.properties` file and the `gfsecurity.properties` file. You can also specify a `cache.xml` file in the cache server's working directory.
- By default, a new server started with `gfsh` receives its initial cache configuration from the cluster configuration service, assuming the locator is running the cluster configuration service. If you specify a group when starting the server, the server also receives configurations that apply to a group. The shared configuration consists of `cache.xml` files, `gemfire.properties` files, and deployed jar files. You can disable use of the cluster configuration service by specifying `--use-cluster-configuration=false` when starting the server using `gfsh`. See Overview of the Cluster Configuration Service.
- If you are using the Spring Framework, you can specify a Spring ApplicationContext XML file when starting up your server in `gfsh` by using the `--spring-xml-location` command-line option. This option allows you to bootstrap your VMware GemFire server process with your Spring application's configuration. See Spring documentation for more information on this file.
- For logging output, log file output defaults to `<server-name>.log` in the cache server's working directory. If you restart a server with the same server name, the existing log file is automatically renamed, for example, `server1-01-01.log` and `server1-02-01.log`. You can modify the level of logging details in this file by specifying a level in the `--log-level` argument when starting up the server.
- By default, the server will start in a subdirectory, named after the server's specified name, under the directory where `gfsh` is executed. This subdirectory is considered the current working directory. You can also specify a different working directory when starting the cache server in `gfsh`.
- By default, a server process that has been shutdown and disconnected due to a network partition event or member unresponsiveness will restart itself and automatically try to reconnect to the existing cluster. See Handling Forced Cache Disconnection Using Autoreconnect for more details.
- You can pass JVM parameters to the server's JVM by using the `--J=-Dproperty.name=value` upon server startup. These parameters can be Java properties or VMware GemFire properties such as `gemfire.jmx-manager`. For example:

  ```
  gfsh>start server --name=server1 --J=-Dgemfire.jmx-manager=true \
  --J=-Dgemfire.jmx-manager-start=true --J=-Dgemfire.http-port=8080
  ```

- We recommend that you do not use the `-XX:+UseCompressedStrings` and `-XX:+UseStringCache` JVM configuration properties when starting up servers. These JVM options can cause issues with data corruption and compatibility.

Start the Server with gfsh

See the `gfsh start server` command reference page for syntax information.

These example `gfsh start server` start commands specify a `cache.xml` file for cache configuration, and use different incoming client connection ports:

```
gfsh>start server --name=server1 \
--cache-xml-file=../ServerConfigs/cache.xml --server-port=40404

gfsh>start server --name=server2 \
--cache-xml-file=../ServerConfigs/cache.xml --server-port=40405
```

The location of the `cache.xml` file and the setting for the client connection port could instead be defined within a `gemfire.properties` file. Then, start the server specifying the `gemfire.properties` file, as in the example command:

```
gfsh>start server --name=server1 \
--properties-file=/home/username/cluster/gemfire.properties
```

To start a server with an embedded JMX Manager:

```
gfsh>start server --name=server2 \
--J=-Dgemfire.jmx-manager=true --J=-Dgemfire.jmx-manager-start=true
```

When both `--max-heap` and `--initial-heap` are specified during server startup, additional GC parameters are specified on your behalf. If you do not want additional default GC properties set, then use the `-Xms` & `-Xmx` JVM options to set just these parameters. See Controlling Heap Use with the Resource Manager for more information. To start a server, providing JVM configuration settings:

```
gfsh>start server --name=server3 \
--J=-Xms80m,-Xmx80m --J=-XX:+UseConcMarkSweepGC,-XX:CMSInitiatingOccupancyFraction=65
```

Start the Server Programmatically

Use the `org.apache.geode.distributed.ServerLauncher` API to start the cache server process inside your code. Use the `ServerLauncher.Builder` class to construct an instance of the `ServerLauncher`, and then use the `start()` method to start the server service. The other methods in the `ServerLauncher` class provide status information about the server and allow you to stop the server.

```
import org.apache.geode.distributed.ServerLauncher;

 public class MyEmbeddedServer {

    public static void main(String[] args){
        ServerLauncher serverLauncher  = new ServerLauncher.Builder()
          .setMemberName("server1")
          .setServerPort(40405)
          .set("jmx-manager", "true")
          .set("jmx-manager-start", "true")
          .build();

        serverLauncher.start();

        System.out.println("Cache server successfully started");
    }
}
```

Check Server Status

Once connected to the cluster in `gfsh`, check the status of a running cache server by providing the server name:

```
gfsh>status server --name=server1
```

If you are not connected to a cluster, you can check the status of a local cache server by providing the process ID or the server's current working directory. For example:

```
gfsh>status server --pid=2484
```

or

```
% gfsh status server --dir=server1
```

If successful, the output provides information as in this sample:

```
% gfsh status server --dir=server4
Server in /home/username/server4 on 192.0.2.0[40404] as server4 is currently online.
Process ID: 49008
Uptime: 2 minutes 4 seconds
VMware GemFire Version: 9.10
Java Version: 1..0_272
Log File: /home/username/server4/server4.log
JVM Arguments:
...
```

Stop Server

When connected to the cluster in `gfsh`, stop a running cache server by providing the server name:

```
gfsh>stop server --name=server1
```

If not connected, you can stop a local cache server by specify the server's current working directory or the process ID. For example:

```
gfsh>stop server --pid=2484
```

or

```
gfsh>stop server --dir=server1
```

You can also use the `gfsh shutdown` command to shut down all cache servers in an orderly fashion. Doing a `shutdown` is the correct approach for systems with persistent regions. See Starting Up and Shutting Down Your System for more details.

Managing System Output Files

VMware GemFire output files are optional and can become quite large. Work with your system administrator to determine where to place them to avoid interfering with other system activities.

VMware GemFire includes several types of optional output files as described below.

- **Log Files**. Comprehensive logging messages to help you confirm system configuration and to debug problems in configuration and code. Configure log file behavior in the `gemfire.properties` file. See Logging.
- **Statistics Archive Files**. Standard statistics for caching and distribution activities, which you can archive on disk. Configure statistics collection and archival in the `gemfire.properties`, `archive-disk-space-limit` and `archive-file-size-limit`. See the Reference.
- **Disk Store Files**. Hold persistent and overflow data from the cache. You can configure regions to persist data to disk for backup purposes or overflow to disk to control memory use. The subscription queues that servers use to send events to clients can be overflowed to disk. Gateway sender queues overflow to disk automatically and can be persisted for high availability. Configure these through the `cache.xml`. See Disk Storage.

Firewall Considerations

You can configure and limit port usage for situations that involve firewalls, for example, between client-server or server-server connections.

- **Firewalls and Connections**

  Be aware of possible connection problems that can result from running a firewall on your machine.

- **Firewalls and Ports**

  Make sure your port settings are configured correctly for firewalls.

Firewalls and Connections

Be aware of possible connection problems that can result from running a firewall on your machine.

VMware GemFire is a network-centric distributed system, so if you have a firewall running on your machine it could cause connection problems. For example, your connections may fail if your firewall places restrictions on inbound or outbound permissions for Java-based sockets. You may need to modify your firewall configuration to permit traffic to Java applications running on your machine. The specific configuration depends on the firewall you are using.

As one example, firewalls may close connections to VMware GemFire due to timeout settings. If a firewall senses no activity in a certain time period, it may close a connection and open a new connection when activity resumes, which can cause some confusion about which connections you have.

For more information on how VMware GemFire client and servers connect, see the following topics:

- How Client/Server Connections Work
- Socket Communication
- Controlling Socket Use

- Setting Socket Buffer Sizes

Firewalls and Ports

Make sure your port settings are configured correctly for firewalls.

There are several different port settings that need to be considered when using firewalls:

- Port that the cache server listens on. This is configurable using the `cache-server` element in cache.xml, on the CacheServer class in Java APIs, and as a command line option to the `gfsh start server` command.

  By default, if not otherwise specified, VMware GemFire clients and servers discover each other on a pre-defined port (**40404**) on the localhost.

- Locator port. VMware GemFire clients can use the locator to automatically discover cache servers. The locator port is configurable as a command-line option to the `gfsh start locator` command. Locators are used in the peer-to-peer cache deployments to discover other processes. They can be used by clients to locate servers as an alternative to configuring clients with a collection of server addresses and ports.

  By default, if not otherwise specified, VMware GemFire locators use the default port **10334**.

- Since locators start up the cluster, locators must also have their ephemeral port range and TCP port accessible to other members through the firewall.

- For clients, you configure the client to connect to servers using the client’s pool configuration. The client’s pool configuration has two options: you can create a pool with either a list of server elements or a list of locator elements. For each element, you specify the host and port. The ports specified must be made accessible through your firewall.

Limiting Ephemeral Ports for Peer-to-Peer Membership

By default, VMware GemFire assigns *ephemeral* ports, that is, temporary ports assigned from a designated range, which can encompass a large number of possible ports. When a firewall is present, the ephemeral port range usually must be limited to a much smaller number, for example six. If you are configuring P2P communications through a firewall, you must also set the TCP port for each process and ensure that UDP traffic is allowed through the firewall.

Properties for Firewall and Port Configuration

This table contains properties potentially involved in firewall behavior, with a brief description of each property.

Configuration area	Property or Setting	Definition
peer-to-peer config	`conserve-sockets`	Specifies whether sockets are shared by the system member's threads.

Configuration area	Property or Setting	Definition
peer-to-peer config	`locators`	The list of locators used by system members. The list must be configured consistently for every member of the cluster.
peer-to-peer config	`mcast-address`	Address used to discover other members of the cluster. Only used if mcast-port is non-zero. This attribute must be consistent across the cluster.
peer-to-peer config	`mcast-port`	Port used, along with the mcast-address, for multicast communication with other members of the cluster. If zero, multicast is disabled for data distribution.
peer-to-peer config	`membership-port-range`	The range of ephemeral ports available for unicast UDP messaging and for TCP failure detection in the peer-to-peer cluster.
peer-to-peer config	`tcp-port`	The TCP port to listen on for cache communications.

Configuration Area	Property or Setting	Definition
cache server config	`hostname-for-clients`	Hostname or IP address to pass to the client as the location where the server is listening.
cache server config	`max-connections`	Maximum number of client connections for the server. When the maximum is reached, the server refuses additional client connections.
cache server config	`port` (cache.xml) or `--port` parameter to the `gfsh start server` command	Port that the server listens on for client communication.

Default Port Configurations

Port Name	Related Configuration Setting	Default Port
Cache Server	`port` (cache.xml)	40404
HTTP	`http-service-port`	7070

Port Name	Related Configuration Setting	Default Port
Locator	`start-locator` (for embedded locators) or `--port` parameter to the `gfsh start locator` command.	*if not specified upon startup or in the start-locator property, uses default port 10334*
Membership Port Range	`membership-port-range`	41000 to 61000
Memcached Port	`memcached-port`	*not set*
Multicast	`mcast-port`	0
RMI	`jmx-manager-port`	1099
TCP	`tcp-port`	ephemeral port

Properties for Firewall and Port Configuration in Multi-Site (WAN) Configurations

Each gateway receiver uses a single port to accept connections from gateway senders in other systems. The configuration of a gateway receiver specifies a range of possible port values to use. VMware GemFire selects an available port from the specified range when the gateway receiver starts. Configure your firewall so that the full range of possible port values is accessible by gateway senders from across the WAN.

Configuration Area	Property or Setting	Definition
multi-site (WAN) config for gateway sender	hostname-for-senders	Hostname or IP address of the gateway receiver used by gateway senders to connect.
multi-site (WAN) config for locator	remote-locators	List of locators (and their ports) that are available on the remote WAN site.
multi-site (WAN) config for gateway receiver	start-port and end-port (cache.xml) or `-start-port` and `-end-port` parameters to the `gfsh start gateway receiver` command	Port range that the gateway receiver can use to listen for gateway sender communication.

Basic Configuration and Programming

Basic Configuration and Programming describes how to configure cluster and cache properties for your VMware GemFire installation. For your applications, it provides guidance for writing code to manage your cache and cluster connection, data regions, and data entries, including custom classes.

- **Cluster and Cache Configuration**

  To work with your VMware GemFire applications, you use a combination of configuration files and application code.

- **Cache Management**

  The VMware GemFire cache is the entry point to VMware GemFire caching management. VMware GemFire provides different APIs and XML configuration models to support the behaviors of different members.

- **Data Regions**

  The region is the core building block of the VMware GemFire cluster. All cached data is organized into data regions and you do all of your data puts, gets, and querying activities against them.

- **Data Entries**

  The data entry is the key/value pair where you store your data. You can manage your entries individually and in batches. To use domain objects for your entry values and keys, you need to follow VMware GemFire requirements for data storage and distribution.

Cluster and Cache Configuration

To work with your VMware GemFire applications, you use a combination of configuration files and application code.

- **Cluster Members**

  Cluster members are programs that connect to a VMware GemFire cluster. You configure members to belong to a single cluster, and you can optionally configure them to be clients or servers to members in other clusters, and to communicate with other clusters.

- **Setting Properties**

  VMware GemFire provides a default cluster configuration for out-of-the-box systems. To use non-default configurations and to fine-tune your member communication, you can use a mix of various options to customize your cluster configuration.

- **Options for Configuring the Cache and Data Regions**

To populate your VMware GemFire cache and fine-tune its storage and distribution behavior, you need to define cached data regions and provide custom configuration for the cache and regions.

- **Local and Remote Membership and Caching**

  For many VMware GemFire discussions, you need to understand the difference between local and remote membership and caching.

Cluster Members

Cluster members are programs that connect to a VMware GemFire cluster. You configure members to belong to a single cluster, and you can optionally configure them to be clients or servers to members in other clusters, and to communicate with other clusters.

Member Overview

Cluster members (or simply “members”) connect to the VMware GemFire cluster when they create the VMware GemFire data cache. The members’ cluster is configured through VMware GemFire properties. See gemfire.properties and gfsecurity.properties (VMware GemFire Properties). VMware GemFire properties specify all necessary information for member startup, initialization, and communication.

Note: You cannot change a member’s properties while the member is connected to the cluster.

Use the properties to define:

- How to find and communicate with other members
- How to perform logging and statistics activities
- Which persistent configuration or `cache.xml` file to use for cache and data region initialization
- Other options, including event conflation, how to handle network loss, and security settings

Membership and System Topologies

Every VMware GemFire process is a member of a cluster, even if the cluster is defined as standalone, with just one member. You can run an individual cluster in isolation or you can combine clusters for vertical and horizontal scaling. See Topology and Communication General Concepts.

- **Peer-to-Peer Clusters**. Members that define the same member discovery properties belong to the same cluster and are peers to one another.
- **Client/Server Installations**. The client/server topology uses relationships that you configure between members of multiple clusters. You configure some or all of the peers in one cluster to act as cache servers to clients connecting from outside the cluster. Each server can host many client processes, managing cache access for all in an efficient, vertically hierarchical cache configuration. You configure the client applications to connect to the servers, using a client cache configuration. Clients run as members of standalone VMware GemFire clusters, with no peers, so all data updates and requests go to the servers.

Multi-site Installations

The multi-site topology uses relationships that you configure between members of multiple clusters. Through this configuration, you loosely couple two or more clusters for automated data distribution. This is usually done for sites at geographically separate locations. You configure a subset of peers in each cluster site with gateway senders and/or gateway receivers to manage events that are distributed between the sites.

In the context of a single cluster, unless otherwise specified, “remote member” refers to another member of the same cluster. In client/server and multi-site installations, “remote” generally refers to members in other clusters. For example, all servers are “remote” to the clients that connect to them. Each client runs standalone, with connections only to the server tier, so all servers and their other clients are “remote” to the individual client. All gateway receivers are “remote” to the gateway senders that connect to them from other clusters, and to those gateway senders’ peers.

Setting Properties

VMware GemFire provides a default configuration for out-of-the-box systems. To use non-default configurations and to fine-tune your member communication, you can use a mix of various options to customize your configuration.

VMware GemFire properties are used to join a cluster and configure system member behavior. Configure your VMware GemFire properties through the `gemfire.properties` file, the Java API, or command-line input. Generally, you store all your properties in the `gemfire.properties` file, but you may need to provide properties through other means, for example, to pass in security properties for a username and password that you have received from keyboard input.

Note: Check with your VMware GemFire system administrator before changing properties through the API, including the `gemfire.properties` and `gfsecurity.properties` settings. The system administrator may need to set properties at the command line or in configuration files. Any change made through the API overrides those other settings.

Note: The product `defaultConfigs` directory has a sample `gemfire.properties` file with all default settings.

Set properties by any combination of the following. The system looks for the settings in the order listed:

1. `java.lang.System` property setting. Usually set at the command line. For applications, set these in your code or at the command line.

   Naming: Specify these properties in the format `gemfire.property-name`, where `property-name` matches the name in the `gemfire.properties` file. To set the gemfire property file name, use `gemfirePropertyFile` by itself - In the API, set the `System` properties before the cache creation call. Example:

   ~~~
   ``` pre
   System.setProperty("gemfirePropertyFile", "gfTest");
   System.setProperty("gemfire.mcast-port", "10999");

   Cache cache = new CacheFactory().create();
   ```
   ~~~

- At the `java` command line, pass in `System` properties using the `-D` switch. Example:

  ```
  java -DgemfirePropertyFile=gfTest -Dgemfire.mcast-port=10999 test.Program
  ```

2. Entry in a `Properties` object.

   Naming: Specify these properties using the names in the `gemfire.properties` file. To set the gemfire property file name, use `gemfirePropertyFile`. - In the API, create a `Properties` object and pass it to the cache create method. Example:

   ~~~
   ``` pre
   Properties properties= new Properties();
   properties.setProperty("log-level", "warning");
   properties.setProperty("name", "testMember2");
   ClientCache userCache =
       new ClientCacheFactory(properties).create();
   ```
   ~~~

   - For the cache server, pass the properties files on the `gfsh` command line as command-line options. Example:

     ```
     gfsh>start server --name=server_name --mcast-port=10338 --properties-file
     =serverConfig/gemfire.properties --security-properties-file=gfsecurity.pr
     operties
     ```

     See Running VMware GemFire Server Processes for more information on running cache servers.

3. Entry in a `gemfire.properties` file. See Deploying Configuration Files without the Cluster Configuration Service. Example:

   ```
   cache-xml-file=cache.xml
   conserve-sockets=true
   disable-tcp=false
   ```

4. Default value. The default values are defined within the API for `org.apache.geode.distributed.ConfigurationProperties`.

Options for Configuring the Cache and Data Regions

To populate your VMware GemFire cache and fine-tune its storage and distribution behavior, you need to define cached data regions and provide custom configuration for the cache and regions.

Cache configuration properties define:

- Cache-wide settings such as disk stores, communication timeouts, and settings designating the member as a server
- Cache data regions

Configure the cache and its data regions through one or more of these methods:

- Through a persistent configuration that you define when issuing commands that use the gfsh command line utility. The gfsh utility supports the administration, debugging, and deployment of VMware GemFire processes and applications. You can use gfsh to configure regions, locators, servers, disk stores, event queues, and other objects.

As you issue commands, gfsh saves a set of configurations that apply to the entire cluster and also saves configurations that only apply to defined groups of members within the cluster. You can re-use these configurations to create a cluster. See Overview of the Cluster Configuration Service.

- Through declarations in the XML file named in the `cache-xml-file` `gemfire.properties` setting. This file is generally referred to as the `cache.xml` file, but it can have any name. See cache.xml.
- Through application calls to the `org.apache.geode.cache.CacheFactory`, `org.apache.geode.cache.Cache` and `org.apache.geode.cache.Region` APIs.

Local and Remote Membership and Caching

For many VMware GemFire discussions, you need to understand the difference between local and remote membership and caching.

Discussions of VMware GemFire membership and caching activities often differentiate between local and remote. Local caching always refers to the central member under discussion, if there is one such obvious member, and remote refers to other members. If there is no clear, single local member, the discussion assigns names to the members to differentiate. Operations, data, configuration, and so forth that are “local to member Q” are running or resident inside the member Q process. Operations, data, configuration, and so on, that are “remote to member Q” are running or resident inside some other member.

The local cache is the cache belonging to the local member. All other caches are remote, whether in other members of the same cluster or in different clusters.

Cache Management

The VMware GemFire cache is the entry point to VMware GemFire caching management. VMware GemFire provides different APIs and XML configuration models to support the behaviors of different members.

- **Introduction to Cache Management**

  The cache provides in-memory storage and management for your data.

- **Managing a Peer or Server Cache**

  You start your peer or server cache using a combination of XML declarations and API calls. Close the cache when you are done.

- **Managing a Client Cache**

  You have several options for client cache configuration. Start your client cache using a combination of XML declarations and API calls. Close the client cache when you are done.

- **Managing a Cache in a Secure System**

  When you create your cache in a secure system, you provide credentials to the connection process for authentication by already-running, secure members. Clients connect to secure servers. Peers are authenticated by secure locators or peer members.

- **Managing RegionServices for Multiple Secure Users**

  In a secure system, you can create clients with multiple, secure connections to the servers from each client. The most common use case is a VMware GemFire client embedded in an application server that supports data requests from many users. Each user may be authorized to access a subset of data on the servers. For example, customer users may be allowed to see and update only their own orders and shipments.

- **Launching an Application after Initializing the Cache**

  You can specify a callback application that is launched after the cache initialization.

Introduction to Cache Management

The cache provides in-memory storage and management for your data.

You organize your data in the cache into *data regions*, each with its own configurable behavior. You store your data into your regions in key/value pairs called *data entries*. The cache also provides features like transactions, data querying, disk storage management, and logging. See the Javadocs for `org.apache.geode.cache.Cache`.

You generally configure caches using the `gfsh` command-line utility or a combination of XML declarations and API calls. VMware GemFire loads and processes your XML declarations when you first create the cache.

VMware GemFire has one cache type for managing server and peer caches and one for managing client caches. The cache server process automatically creates its server cache at startup. In your application process, the cache creation returns an instance of the server/peer or client cache. From that point on, you manage the cache through API calls in your application.

The Caching APIs

VMware GemFire's caching APIs provide specialized behavior for different system member types and security settings.

- `org.apache.geode.cache.RegionService`. Generally, you use the `RegionService` functionality through instances of `Cache` and `ClientCache`. You only specifically use instances of `RegionService` for limited-access users in secure client applications that service many users. The `RegionService` API provides access to existing cache data regions and to the standard query service for the cache. For client caches, queries are sent to the server tier. For server and peer caches, queries are run in the current cache and any available peers. `RegionService` is implemented by `GemFireCache`.
- `org.apache.geode.cache.GemFireCache`. You do not specifically use instances of `GemFireCache`, but you use `GemFireCache` functionality in your instances of `Cache` and `ClientCache`. `GemFireCache` extends `RegionService` and adds general caching features like region attributes, disk stores for region persistence and overflow, and access to the underlying cluster. `GemFireCache` is implemented by `Cache` and `ClientCache`.
- `org.apache.geode.cache.Cache`. Use the `Cache` interface to manage server and peer caches. You have one `Cache` per server or peer process. The `Cache` extends `GemFireCache` and adds

server/peer caching features like communication within the cluster, region creation, transactions and querying, and cache server functionality.

- org.apache.geode≈setting_cache_initializer.cache.ClientCache. Use the ClientCache interface to manage the cache in your clients. You have one ClientCache per client process. The ClientCache extends GemFireCache and adds client-specific caching features like client region creation, subscription keep-alive management for durable clients, querying on server and client tiers, and RegionService creation for secure access by multiple users within the client.

The Cache XML

Your cache.xml must be formatted according to the product XML schema definition cache-1.0.xsd. The schema definition file is available at http://geode.apache.org/schema/cache/cache-1.0.xsd.

You use one format for peer and server caches and another for client caches.

cache.xml for Peer/Server:

```
<?xml version="1.0" encoding="UTF-8"?>
<cache xmlns="http://geode.apache.org/schema/cache"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://geode.apache.org/schema/cache http://geode.apache.org/s
chema/cache/cache-1.0.xsd"
    version="1.0”>
...
</cache>
```

cache.xml for Client:

```
<?xml version="1.0" encoding="UTF-8"?>
<client-cache
    xmlns="http://geode.apache.org/schema/cache"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://geode.apache.org/schema/cache http://geode.apache.org/s
chema/cache/cache-1.0.xsd"
    version="1.0”>
...
</client-cache>
```

For more information on the cache.xml file, see cache.xml.

Create and Close a Cache

Your system configuration and cache configuration are initialized when you start your member processes and create each member’s VMware GemFire cache. If you are using the cluster configuration service, member processes can pick up its cache configuration from the cluster or group’s current configuration. See Overview of the Cluster Configuration Service.

The steps in this section use gemfire.properties and cache.xml file examples, except where API is required. You can configure your cluster properties and cache through the API as well, and you can use a combination of file configuration and API configuration.

The XML examples may not include the full `cache.xml` file listing. All of your declarative cache configuration must conform to the cache XSD at http://geode.apache.org/schema/cache/cache-1.0.xsd.

For all of your VMware GemFire applications:

1. Create your `Cache`, for peer/server applications, or `ClientCache`, for client applications. This connects to the VMware GemFire system you have configured and initializes any configured data regions. Use your cache instance to access your regions and perform your application work.
2. Close your cache when you are done. This frees up resources and disconnects your application from the cluster in an orderly manner.

Follow the instructions in the subtopics under Cache Management to customize your cache creation and closure for your application needs. You may need to combine more than one of the sets of instructions. For example, to create a client cache in a system with security, you would follow the instructions for creating and closing a client cache and for creating and closing a cache in a secure system.

Export and Import a Cache Snapshot

To aid in the administration of cache data and speed the setup of new environments, you can export a snapshot of the entire cache (all regions) and then import the snapshot into a new cache. For example, you could take a snapshot of the production environment cache in order to import the cache's data into a testing environment.

For more details on exporting and importing snapshots of a cache, see Cache and Region Snapshots.

Cache Management with gfsh and the Cluster Configuration Service

You can use gfsh commands to mange a server cache. There are gfsh commands to create regions, start servers, and to create queues and other objects. As you issue these commands, the Cluster Configuration Service saves cache.xml and gemfire.properties files on the locators and distributes those configurations to any new members that join the cluster. See Overview of the Cluster Configuration Service.

Managing a Peer or Server Cache

You start your peer or server cache using a combination of XML declarations and API calls. Close the cache when you are done.

VMware GemFire peers are members of a VMware GemFire cluster that do not act as clients to another VMware GemFire cluster. VMware GemFire servers are peers that also listen for and process client requests.

1. Create your cache:
    1. Start up a cluster and the cluster configuration service:

1. Start a locator with `--enable-cluster-configuration` set to true. (It is set true by default.)

   ```
   gfsh>start locator --name=locator1
   ```

2. Start up member processes that use the cluster configuration service (enabled by default):

   ```
   gfsh>start server --name=server1 --server-port=40404
   ```

3. Create regions:

   ```
   gfsh>create region --name=customerRegion --type=REPLICATE

   gfsh>create region --name=ordersRegion --type=PARTITION
   ```

2. Or if you are not using the cluster configuration service, directly configure cache.xml in each member of your cluster. In your `cache.xml`, use the `cache` DOCTYPE and configure your cache inside a `<cache>` element. Example:

```
<?xml version="1.0" encoding="UTF-8"?>
<cache
    xmlns="http://geode.apache.org/schema/cache"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://geode.apache.org/schema/cache http://geod
e.apache.org/schema/cache/cache-1.0.xsd"
    version="1.0”>
    // NOTE: Use this <cache-server> element only for server processes
    <cache-server port="40404"/>
    <region name="customerRegion" refid="REPLICATE" />
    <region name="ordersRegion" refid="PARTITION" />
</cache>
```

3. To programmatically create the `Cache` instance:

   - In your Java application, use the `CacheFactory` create method:

     ```
     Cache cache = new CacheFactory().create();
     ```

   - If you are running a server using the VMware GemFire `cacheserver` process, it automatically creates the cache and connection at startup and closes both when it exits.

   The system creates the connection and initializes the cache according to your `gemfire.properties` and `cache.xml` specifications.

2. Close your cache when you are done using the inherited `close` method of the `Cache` instance:

```
cache.close();
```

Managing a Client Cache

You have several options for client cache configuration. Start your client cache using a combination of XML declarations and API calls. Close the client cache when you are done.

VMware GemFire clients are processes that send most or all of their data requests and updates to a VMware GemFire server system. Clients run as standalone processes, without peers of their own.

Note: VMware GemFire automatically configures the cluster for your `ClientCache` as standalone, which means the client has no peers. Do not try to set the `gemfire.properties` `mcast-port` or `locators` for a client application or the system will throw an exception.

1. Create your client cache:
    1. In your `cache.xml`, use the `client-cache` DOCTYPE and configure your cache inside a `<client-cache>` element. Configure your server connection pool and your regions as needed. Example:

        ```
        <?xml version="1.0" encoding="UTF-8"?>
        <client-cache
            xmlns="http://geode.apache.org/schema/cache"
            xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            xsi:schemaLocation="http://geode.apache.org/schema/cache http://geod
        e.apache.org/schema/cache/cache-1.0.xsd"
            version="1.0">
            <pool name="serverPool">
                <locator host="host1" port="44444"/>
            </pool>
            <region name="exampleRegion" refid="PROXY"/>
        </client-cache>
        ```

        Note: Applications that use a `client-cache` may want to set `concurrency-checks-enabled` to false for a region in order to see all events for that region. VMware GemFire server members can continue using concurrency checks, but they will pass all events to the client cache. This configuration ensures that the client sees all region events, but it does not prevent the client cache region from becoming out-of-sync with the server cache. See Consistency for Region Updates.

    2. If you use multiple server pools, configure the pool name explicitly for each client region. Example:

        ```
        <pool name="svrPool1">
            <locator host="host1" port="40404"/>
        </pool>
        <pool name="svrPool2">
            <locator host="host2" port="40404"/>
        </pool>
        <region name="clientR1" refid="PROXY" pool-name="svrPool1"/>
        <region name="clientR2" refid="PROXY" pool-name="svrPool2"/>
        <region name="clientsPrivateR" refid="LOCAL"/>
        ```

    3. In your Java client application, create the cache using the `ClientCacheFactory` `create` method. Example:

        ```
        ClientCache clientCache = new ClientCacheFactory().create();
        ```

This creates the server connections and initializes the client's cache according to your `gemfire.properties` and `cache.xml` specifications.

2. Close your cache when you are done using the `close` method of your `Cache` instance:

```
cache.close();
```

If your client is durable and you want to maintain your durable queues while the client cache is closed, use:

```
clientCache.close(true);
```

Managing a Cache in a Secure System

A secured system does both authentication at connection time and authorization prior to cache operations. Client apps and cluster members (servers and locators) require configuration and setup when the `SecurityManager` is enabled.

See the section on Security for details. For authentication, see Implementing Authentication.

Managing RegionServices for Multiple Secure Users

In a secure system, you can create clients with multiple, secure connections to the servers from each client. The most common use case is a VMware GemFire client embedded in an application server that supports data requests from many users. Each user may be authorized to access a subset of data on the servers. For example, customer users may be allowed to see and update only their own orders and shipments.

In a single client, multiple authenticated users can all access the same `ClientCache` through instances of the `RegionService` interface. Because there are multiple users with varying authorization levels, access to cached data is done entirely through the servers, where each user's authorization can be managed. Follow these steps in addition to the steps in Managing a Cache in a Secure System.

1. Create your cache and `RegionService` instances:

    1. Configure your client's server pool for multiple secure user authentication. Example:

        ```
        <pool name="serverPool" multiuser-authentication="true">
            <locator host="host1" port="44444"/>
            </pool>
        ```

        This enables access through the pool for the `RegionService` instances and disables it for the `ClientCache` instance.

    2. After you create your `ClientCache`, from your `ClientCache` instance, for each user call the `createAuthenticatedView` method, providing the user's particular credentials. These are create method calls for two users:

        ```
        Properties properties = new Properties();
        properties.setProperty("security-username", cust1Name);
        properties.setProperty("security-password", cust1Pwd);
        RegionService regionService1 =
        ```

```
    clientCache.createAuthenticatedView(properties);

properties = new Properties();
properties.setProperty("security-username", cust2Name);
properties.setProperty("security-password", cust2Pwd);
RegionService regionService2 =
    clientCache.createAuthenticatedView(properties);
```

   For each user, do all of your caching and region work through the assigned `RegionService` instance. Access to the server cache will be governed by the server's configured authorization rules for each individual user.

2. Close your cache by closing the `ClientCache` instance only. Do not close the `RegionService` instances first. This is especially important for durable clients.

Requirements and Caveats for RegionService

Once each region is created, you can perform operations on it through the `ClientCache` instance or the `RegionService` instances, but not both.

Note: You can use the `ClientCache` to create a region that uses a pool configured for multi-user authentication, then access and do work on the region using your `RegionService` instances.

To use `RegionService`, regions must be configured as `EMPTY`. Depending on your data access requirements, this configuration might affect performance, because the client goes to the server for every get.

Launching an Application after Initializing the Cache

You can specify a callback application that is launched after the cache initialization.

By specifying an `<initializer>` element in your cache.xml file, you can trigger a callback application, which is run after the cache has been initialized. Applications that use the cacheserver script to start up a server can also use this feature to hook into a callback application. To use this feature, you need to specify the callback class within the `<initializer>` element. This element should be added to the end of your `cache.xml` file.

You can specify the `<initializer>` element for either server caches or client caches.

The callback class must implement the `Declarable` interface. When the callback class is loaded, its `init` method is called, and any parameters defined in the `<initializer>` element are passed as properties.

The following is an example specification.

In cache.xml:

```
<initializer>
   <class-name>MyInitializer</class-name>
      <parameter name="members">
         <string>2</string>
      </parameter>
</initializer>
```

Here's the corresponding class definition:

```
import org.apache.geode.cache.Declarable;

public class MyInitializer implements Declarable {
   public void init(Properties properties) {
      System.out.println(properties.getProperty("members"));
   }
}
```

The following are some additional real-world usage scenarios:

1. Start a SystemMembershipListener

   ```
   <initializer>
      <class-name>TestSystemMembershipListener</class-name>
   </initializer>
   ```

2. Write a custom tool that monitors cache resources

   ```
   <initializer>
      <class-name>ResourceMonitorCacheXmlLoader</class-name>
   </initializer>
   ```

Any singleton or timer task or thread can be instantiated and started using the initializer element.

Data Regions

The region is the core building block of the VMware GemFire cluster. All cached data is organized into data regions and you do all of your data puts, gets, and querying activities against them.

- **Region Management**

  VMware GemFire provides `gfsh` commands, APIs, and XML configuration models to support the configuration and management of data regions.

- **Region Naming**

  To be able to perform all available operations on your data regions, follow these region naming guidelines.

- **Region Shortcuts and Custom Named Region Attributes**

  VMware GemFire provides region shortcut settings, with preset region configurations for the most common region types. For the easiest configuration, start with a shortcut setting and customize as needed. You can also store your own custom configurations in the cache for use by multiple regions.

- **Storing and Retrieving Region Shortcuts and Custom Named Region Attributes**

  Use these examples to get started with VMware GemFire region shortcuts.

- **Managing Region Attributes**

  Use region attributes to fine-tune the region configuration provided by the region shortcut settings.

- **Creating Custom Attributes for Regions and Entries**

Use custom attributes to store information related to your region or its entries in your cache. These attributes are only visible to the local application and are not distributed.

- **Building a New Region with Existing Content**

  A new region or cluster may need to be loaded with the data of an existing system. There are two approaches to accomplish this task. The approach used depends upon the organization of both the new and the existing cluster.

Region Management

Operations that create, destroy, invalidate, clear, and change the configuration of regions work with gfsh commands, through an XML description, and via API calls.

You store your data in region entry key/value pairs, with keys and values being any object types your application needs. The `org.apache.geode.cache.Region` interface implements `java.util.Map`.

Each region's attributes define how the data in the region is stored, distributed, and managed. Data regions can be distributed, partitioned among system members, or local to the member.

Region shortcuts identify commonly-used types of regions. See Region Shortcuts for more information.

Note: If you change attributes that define a region, you must restart the member for the changes to take effect.

Creating a Region

Creating a Region with gfsh

A simple and fast way to create a data region in the VMware GemFire cache is to use the `gfsh` command-line tool.

Region creation is subject to attribute consistency checks, both internal to the cache and, if the region is not local, between all caches where the region is defined.

The `gfsh create region` command reference page details command line options for creating a region with `gfsh`.

With `gfsh` connected to a JMX server, an example command that creates a replicated region is

```
gfsh>create region --name=region1 --type=REPLICATE
```

Export the configuration files of your server so that you can save your region's configuration and recreate the region with the same attributes the next time you start up your cache server. See export config for details.

Note: The cluster configuration service, which is enabled by default, automatically saves the configuration on the locators in the cluster. After you use the gfsh create region command, any new servers that you start that attach to the same locator receive the same configuration. You can also create alternate configurations within a cluster by specifying a group when creating the region and starting servers. See Overview of the Cluster Configuration Service.

Creating a Region Through the cache.xml File

A common way to create a data region in the VMware GemFire cache is through `cache.xml` declarations. When starting the member with the `cache.xml` file, the region will be created.

Region creation is subject to attribute consistency checks, both internal to the cache and, if the region is not local, between all caches where the region is defined.

- In the `cache.xml` file, create a `<region>` element for the new region as a subelement to the `<cache>` element or the `<client-cache>` element.
- Define the region's name and use a region shortcut, if one applies.
- Add other attributes as needed to customize the region's behavior.

cache.xml File Examples

The `region` declaration of a replicated region named Portfolios:

```
<region name="Portfolios" refid="REPLICATE"/>
```

The `region` declaration of a partitioned region named myRegion:

```
<region name="myRegion" refid="PARTITION"/>
```

The `region` declaration of a partitioned region that backs up content to disk:

```
<region name="myRegion" refid="PARTITION_PERSISTENT"/>
```

The `region` declaration of a partitioned region configured with high availability and a modified storage capacity in the host member:

```
<region name="myRegion" refid="PARTITION_REDUNDANT">
    <region-attributes>
        <partition-attributes local-max-memory="512" />
    </region-attributes>
</region>
```

The `region` declaration of a replicated region configured with an event listener in which entries expire:

```
<region name="myRegion" refid="REPLICATE">
    <region-attributes statistics-enabled="true">
        <entry-time-to-live>
            <expiration-attributes timeout="60" action="destroy"/>
        </entry-time-to-live>
        <cache-listener>
            <class-name>myPackage.MyCacheListener</class-name>
        </cache-listener>
    </region-attributes>
</region>
```

Creating a Region Through the API

VMware GemFire's regions APIs provide specialized behavior for different system member types.

- **Peer/Server Region APIs**. Use these methods, interfaces, and classes for peer/server region creation. These are in the `org.apache.geode.cache` package. They correspond to `cache.xml` declarations within the `<cache>` element for creating and configuring regions.
    - `org.apache.geode.cache.Cache.createRegionFactory` . This method takes a `RegionShortcut enum` to initiate region configuration, and it returns a `RegionFactory`. Use `createRegionFactory()`, not `new RegionFactory`, to create a RegionFactory.
    - `org.apache.geode.cache.RegionFactory`. Provides methods to set individual region attributes and to create the region. The `create` call returns a `Region`.
    - `org.apache.geode.cache.RegionShortcut`. Defines common region configurations.
- **Client Region APIs**. Use these methods, interfaces, and classes for client region creation. These are in the `org.apache.geode.cache.client` package. They correspond to `cache.xml` declarations in the `<client-cache>` element for creating and configuring regions.

  These are client versions of the Peer/Server Region APIs. These client APIs provide similar functionality, but are tailored to the needs and behaviors of client regions.

    - `org.apache.geode.cache.clientCache.createRegionFactory` . This method takes a `ClientRegionShortcut enum` to initiate region configuration, and returns a `ClientRegionFactory`.
    - `org.apache.geode.cache.client.ClientRegionFactory`. Provides methods to set individual region attributes and to create the region. The `create` call returns `Region`.
    - `org.apache.geode.cache.client.ClientRegionShortcut` . Defines common region configurations.
- **Region APIs Used For All Member Types**. These interfaces and classes are used universally for region management. These are in the `org.apache.geode.cache` package. They correspond to `cache.xml` declarations in the `<cache>` and `<client-cache>` elements for creating and configuring regions.
    - `org.apache.geode.cache.Region` . Interface for managing regions and their entries.
    - `org.apache.geode.cache.RegionAttributes` . Object holding region configuration settings.

Use the API to create regions in the cache after startup. For run-time region creation, you need to use the API.

Region creation is subject to attribute consistency checks, both internal to the cache and, if the region is not local, between all caches where the region is defined.

1. Use a region shortcut to create your region factory.
    - In peers and servers, use `org.apache.geode.cache.RegionFactory`.
    - In clients, use `org.apache.geode.cache.client.ClientRegionFactory`.
2. (Optional) Use the region factory to further configure your region.
3. Create your region from the configured region factory.

API Examples

Create a replicated region named Portfolios:

```
Cache cache = CacheFactory.create();
RegionFactory rf = cache.createRegionFactory(REPLICATE);
Region pfloRegion = rf.create("Portfolios");
```

Create a partitioned region with a listener:

```
RegionFactory rf =
    cache.createRegionFactory(RegionShortcut.PARTITION);
rf.addCacheListener(new LoggingCacheListener());
custRegion = rf.create("customer");
```

Create a partitioned region with a partition resolver for colocated regions:

```
PartitionAttributesFactory paf = new PartitionAttributesFactory<CustomerId, String>();
paf.setPartitionResolver(new CustomerOrderResolver());

RegionFactory rf =
    cache.createRegionFactory(RegionShortcut.PARTITION);
rf.setPartitionAttributes(paf.create());
rf.addCacheListener(new LoggingCacheListener());
custRegion = rf.create("customer");
```

Create a client region with a pool specification:

```
ClientRegionFactory<String,String> cRegionFactory =
    cache.createClientRegionFactory(PROXY);
Region<String, String> region =
    cRegionFactory.setPoolName("Pool3").create("DATA");
```

Create and Access Data Subregions

An individual region can contain multiple subregions. Subregions are an older feature that will not be useful in new designs and applications. They are used to create a hierarchical namespace within a cache, providing naming that feels like paths in a file system. Here are limitations on the use of subregions:

- A region with LOCAL scope can only have subregions with LOCAL scope.
- Partitioned region types may not be used with subregions. A subregion may not have a parent that is a partitioned region, and a subregion may not be of type PARTITION.
- A subregion must have the same scope (GLOBAL, DISTRIBUTED_ACK, DISTRIBUTED_NO_ACK) as its parent region.
- Subregion names must be unique within the cache.

You can create subregions using one of the following methods:

- Declaration in the `cache.xml`:

  ```
  <?xml version="1.0"?>
  <cache
      xmlns="http://geode.apache.org/schema/cache"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://geode.apache.org/schema/cache http://geode.apach
  e.org/schema/cache/cache-1.0.xsd"
  ```

```
    version="1.0"
    lock-lease="120"
    lock-timeout="60"
    search-timeout="300">
<!-- Create a region named Portfolios -->
  <region name="Portfolios" refid="REPLICATE">
      <region name="Private" refid="REPLICATE">
      ...
      </region>
  </region>
</cache>
```

  When the `cache.xml` is loaded at cache creation, the system automatically creates any declared regions and subregions.

- `RegionFactory` API calls:

  ```
  Cache cache = CacheFactory.create();
  RegionFactory rf = cache.createRegionFactory(REPLICATE);
  Region pfloRegion = rf.create("Portfolios");
  Region pvtSubregion = rf.createSubregion(pfloRegion, "Private");
  ```

`Region` method calls with a `recursive` parameter operate on the given region(s) and then recursively on all contained subregions.

Update the Configuration of Data Regions

Update your region properties and contents through `alter region` command, the API or from `cache.xml` file declarations.

- Use the gfsh alter region command.
- In the API, use `Cache` and `Region` methods to change configuration parameters and modify region structure and data.
- Load new XML declarations using the `Cache.loadCacheXml` method. Where possible, declarations in the new `cache.xml` file supersede existing definitions. For example, if a region declared in the `cache.xml` file already exists in the cache, its mutable attributes are modified according to the file declarations. Immutable attributes are not affected. If a region does not already exist, it is created. Entries and indexes are created or updated according to the state of the cache and the file declarations.

Invalidate a Region

An invalidate region operation removes all entry values for a region, while leaving the entry keys intact. This operation can be invoked only through the API on a `Region` instance. Event notification occurs.

```
// Invalidate the entire distributed region
Region.invalidateRegion();
```

The API also offers a method to invalidate only the entries within the local cache. This method may not be used on a replicated region, as doing so would invalidate the replication contract.

```
// Invalidate the region within this member
Region.localInvalidateRegion();
```

Clear a Region

A clear region operation removes all entries from a region. This operation is not available for partitioned regions. This operation can be invoked through the API on a `Region` instance:

```
// Remove all entries for the region
Region.clear();
```

It can be invoked with the `gfsh` command:

```
gfsh>remove --region=Region1 --all
```

Event notification occurs for a clear region operation.

Destroy a Region

A destroy region operation removes the entire region. This operation can be invoked through the API on a `Region` instance:

```
// Remove the entire region
Region.destroyRegion();
```

A destroy region operation can be invoked with the `gfsh` command:

```
gfsh>destroy region --name=Region1
```

Event notification occurs for a destroy region operation.

A region can be destroyed by removing the region's specification from the `cache.xml` file.

Destroying the region by an API invocation or by using the `gfsh destroy` command while all members are online is the best way to remove a region, as VMware GemFire handles all aspects of the removal, including removing the region's persistent disk stores across the online members hosting the region. Destroying the region by removing its specification from the `cache.xml` file does not remove the region's existing persistent disk stores.

The destroy operation can be propagated only to online members. The system will encounter restart issues if a region is destroyed while some members are online and others are offline. As those members that were offline restart, they will block indefinitely, waiting for persistent region data that no longer exists. To fix this issue, shut down all members that are blocked waiting for the removed region. Once those members are in the offline state, use the `gfsh alter disk-store` command with the `--remove` option on each offline member to remove the region. Then, restart each member.

An edge case results in issues when destroying a persistent region (R-removed) by removing its specification from the `cache.xml` file, and region R-removed was colocated with another persistent region (R-remains). The issue occurs because the persistent information contained within R-remains is inconsistent with the (lack of) specification of R-removed. Upon restart of R-remains, its persisted metadata refers to R-removed as a colocated region, and the startup of R-remains is dependent on that removed region. Thus, the startup of R-remains blocks, unable to complete. The issue may manifest with operations on the R-remains region such as a query, put, or get, that never finishes. To fix this issue, shut down all members with the persisted metadata that refers to the removed region. Once those members are in the offline state, use the `gfsh alter disk-store` command with the `--remove` option on each offline member to remove the region. Then, restart each member.

Close a Region

Use this to stop local caching of persistent and partitioned regions without closing the entire cache:

```
Region.close();
```

The `Region.close` operation works like the `Region.localDestroyRegion` operation with these significant differences:

- The `close` method is called for every callback installed on the region.
- No events are invoked. Of particular note, the entry events, `beforeDestroy` and `afterDestroy`, and the region events, `beforeRegionDestroy` and `afterRegionDestroy`, are not invoked. See Events and Event Handling.
- If persistent, the region is removed from memory but its disk files are retained.
- If partitioned, the region is removed from the local cache. If the partitioned region is redundant, local data caching fails over to another cache. Otherwise, local data is lost.

Region Naming

To be able to perform all available operations on your data regions, follow these region naming guidelines.

- Characters permitted in region names are alphanumeric characters (`ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789`), period (`.`), underscore (`_`), square brackets (`[ ]`), hyphen (`-`), caret (`^`) and backquote (`` ` ``).
- Region names are case sensitive.
- Do not use the slash character (`/`).
- Do not begin region names with two underscore characters (`__`), as this is reserved for internal use.

Region Shortcuts and Custom Named Region Attributes

VMware GemFire provides region shortcut settings, with preset region configurations for the most common region types. For the easiest configuration, start with a shortcut setting and customize as needed. You can also store your own custom configurations in the cache for use by multiple regions.

You configure automated management of data regions and their entries through region shortcuts and region attributes. These region configuration settings determine such things as where the data resides, how the region is managed in memory, reliability behavior, and the automatic loading, distribution, and expiration of data entries.

Note: Whenever possible, use region shortcuts to configure your region, and further customize behavior using region attributes. The shortcut settings are preset with the most common region configurations.

VMware GemFire provides a number of predefined, shortcut region attributes settings for your use. You can also define your own custom region attributes and store them with an identifier for later retrieval. Both types of stored attributes are referred to as named region attributes. You can create and store your attribute settings in the `cache.xml` file and through the API.

Retrieve region shortcuts and custom named attributes by providing the ID to the region creation, in the `refid` attribute setting. This example uses the shortcut REPLICATE attributes to create a region:

```
<region name="testREP" refid="REPLICATE"/>
```

You can create your own named attributes as needed, by providing an `id` in your region attributes declaration. The following region declaration:

1. Retrieves all of the attribute settings provided by the persistent partitioned region shortcut
2. Modifies the shortcut attribute settings by specifying a disk store name to use for persistence
3. Assigns the new attribute settings to the new region named `testPR`
4. Stores the attribute settings in a new custom attributes named `testPRPersist`:

```
<disk-store name="testDiskStore" >
    <disk-dirs>
        <disk-dir>PRPersist1</disk-dir>
        <disk-dir>PRPersist2</disk-dir>
    </disk-dirs>
</disk-store>
<region name="testPR" >
    <region-attributes id="testPRPersist"
        refid="PARTITION_PERSISTENT" disk-store-name="testDiskStore"/>
</region>
```

Shortcut Attribute Options

You can select the most common region attributes settings from VMware GemFire's predefined named region attributes in these classes:

- `org.apache.geode.cache.RegionShortcut`. For peers and servers.
- `org.apache.geode.cache.client.ClientRegionShortcut`. For clients.

Shortcut attributes are a convenience only. They are just named attributes that VMware GemFire has already stored for you. You can override their settings by storing new attributes with the same id as the predefined attributes.

For a full list of all available region shortcuts, see Region Shortcuts Quick Reference.

The `org.apache.geode.cache.RegionShortcut` Javadocs give complete listings of the options.

RegionShortcuts for Peers and Servers

These are the primary options available in the region shortcut settings. The names listed appear in the shortcut identifier alone or in combination, like "`PARTITION`" in `PARTITION`, `PARTITION_PROXY`, and `PARTITION_REDUNDANT`.

Cache Data Storage Mode

- `PARTITION` . Creates a partitioned region. This is a data store for the region. You can also specify these options with `PARTITION`:
    - `PROXY`. Data is not stored in the local cache and the member is a data accessor to the region. This requires other members to create non-proxy copies of the region, so the data is stored somewhere.
    - `REDUNDANT`. The region stores a secondary copy of all data, for high availability.
- `REPLICATE`. Creates a replicated region. This is a data store for the region. You can also specify these options with `REPLICATE`:
    - `PROXY`. Data is not stored in the local cache and the member is a data accessor to the region. This requires other members to create non-proxy copies of the region, so the data is stored somewhere.
- `LOCAL`. Creates a region private to the defining member.

Data Eviction

- `HEAP_LRU`. Causes least recently used data to be evicted from memory when the VMware GemFire resource manager determines that the cache has reached configured storage limits.

Disk Storage

You can specify these alone or in combination:

- `PERSISTENT`. Backs up all data to disk, in addition to storing it in memory.
- `OVERFLOW`. Moves data out of memory and on to disk, when memory use becomes too high.

ClientRegionShortcuts for Clients

These are the primary options available in the client region shortcut settings. The names listed appear in the shortcut identifier alone or in combination, like "`PROXY`" in `PROXY` and `CACHING_PROXY`.

Communication with Servers and Data Storage

- `PROXY`. Does not store data in the client cache, but connects the region to the servers for data requests and updates, interest registrations, and so on. The client is a data accessor to the region.
- `CACHING_PROXY`. Stores data in the client cache and connects the region to the servers for data requests and updates, interest registrations, and so on.
- `LOCAL`. Stores data in the client cache and does not connect the region to the servers. This is a client-side-only region. Note that this is not the same as setting the region's `scope` attribute to `LOCAL`.

Data Eviction

- `HEAP_LRU`. Causes least recently used data to be evicted from memory when the VMware GemFire resource manager determines that the cache has reached configured storage limits.

Disk Storage

With the LOCAL and CACHING data storage shortcut options, you can also specify these disk storage options, alone or in combination:

- `PERSISTENT`. Backs up all data to disk, in addition to storing it in memory.
- `OVERFLOW`. Moves data out of memory and on to disk, when memory use becomes too high.

Storing and Retrieving Region Shortcuts and Custom Named Region Attributes

Use these examples to get started with VMware GemFire region shortcuts.

VMware GemFire region shortcuts, in `org.apache.geode.cache.RegionShortcut` for peers and servers and `org.apache.geode.cache.client.ClientRegionShortcut` for clients, are available wherever you create a region in the `cache.xml` or through the API. Custom named attributes, stored by you, are available from the moment you store them on.

The region shortcuts are special VMware GemFire named region attributes, with identifying names. Create custom named region attributes by setting the attributes and storing them with a unique identifier in the region attribute `id`. Retrieve named attributes by providing the shortcut enum value or the name you assigned in the `id` to the region creation:

- In the API, use the identifier in the region factory creation
- In the `cache.xml`, use the identifier in the `<region>` or `<region-attribute> refid` setting. The `refid` is available in both elements for convenience

Examples

Example #1

This example shows partitioned region creation in the `cache.xml`:

- The first `region-attributes` declaration starts with the predefined `PARTITION_REDUNDANT` attributes, modifies the `local-max-memory` setting, and stores the resulting attributes in the custom-named `myPartition` attributes.
- The region declarations use the new stored attributes, but each has its own interest policy, which is specified in the individual region creation.

  ```
  <!-- Retrieving and storing attributes -->
  <region-attributes id="myPartition" refid="PARTITION_REDUNDANT">
      <partition-attributes local-max-memory="512"/>
  </region-attributes>

  <!-- Two partitioned regions, one colocated with the other -->

  <!-- Attributes are retrieved and applied in the first region -->
  <region name="PartitionedRegion1" refid="myPartition"/>
  ```

```
<!-- Same stored attributes, modification for this region-->
<region name="PartitionedRegion2" refid="myPartition">
    <region-attributes>
        <partition-attributes colocated-with="PartitionedRegion1" />
    </region-attributes>
</region>
```

Example #2

This example uses the `RegionFactory` API to create a region based on the predefined `PARTITION` region shortcut:

```
final Region diskPortfolios =
    new RegionFactory("PARTITION").create("Portfolios");
```

This example retrieves an attributes template and passes it to the region creation with a modified pool specification:

```
ClientRegionFactory<String,String> regionFactory =
    cache.createClientRegionFactory(PROXY);
Region<String, String> region = regionFactory
    .setPoolName("publisher")
    .create("DATA");
```

Managing Region Attributes

Use region attributes to fine-tune the region configuration provided by the region shortcut settings.

All region attributes have default settings, so you only need to use region attributes to set the ones you want to override. See <region-attributes>.

Define Region Attributes

Create region attributes using any of these methods:

- Declarations inside the `cache.xml` `<region>` element:

  ```
  <cache>
     <region name="exampleRegion" refid="REPLICATE">
        <region-attributes statistics-enabled="true">
          <entry-idle-time>
            <expiration-attributes timeout="10" action="destroy"/>
          </entry-idle-time>
          <cache-listener>
            <class-name>quickstart.SimpleCacheListener</class-name>
          </cache-listener>
        </region-attributes>
      </region>
  </cache>
  ```

  When the `cache.xml` is loaded at startup, declared region attributes are automatically created and applied to the region.

- `RegionFactory` API `set`* method calls:

```
// Creating a partitioned region using the RegionFactory
RegionFactory rf = cache.createRegionFactory(RegionShortcut.PARTITION);
rf.addCacheListener(new LoggingCacheListener());
custRegion = rf.create("customer");
```

```
// Creating a partitioned region using the RegionFactory, with attribute modifi
cations
RegionFactory rf =
  cache.createRegionFactory(RegionShortcut.PARTITION);
rf.setPartitionResolver(new CustomerOrderResolver());
rf.addCacheListener(new LoggingCacheListener());
custRegion = rf.create("customer");
```

```
// Creating a client with a Pool Specification Using ClientRegionFactory
ClientRegionFactory<String,String> cRegionFactory =
    cache.createClientRegionFactory(PROXY);
Region<String, String> region =
    cRegionFactory.setPoolName("Pool3").create("DATA");
```

- By issuing the gfsh `create region` command.

Modify Region Attributes

You can modify a region's event handlers and expiration and eviction attributes after the region is created.

Note: Do not modify attributes for existing regions unless absolutely necessary. Creating the attributes you need at region creation is more efficient.

Modify attributes in one of these ways:

- By loading a `cache.xml` with modified region attribute specifications:

```
<!-- Change the listener for exampleRegion
...
    <region name="exampleRegion">
      <region-attributes statistics-enabled="true">
        <cache-listener>
          <class-name>quickstart.ComplicatedCacheListener</class-name>
        </cache-listener>
      </region-attributes>
    </region>
...
```

- Using the `AttributesMutator` API:
  1. Retrieve the `AttributesMutator` from the region
  2. Call the mutator set methods to modify attributes:

```
currRegion = cache.getRegion("root");
AttributesMutator mutator = this.currRegion.getAttributesMutator();
mutator.addCacheListener(new LoggingCacheListener());
```

- By issuing the gfsh `alter region` command. See alter region.

Creating Custom Attributes for Regions and Entries

Use custom attributes to store information related to your region or its entries in your cache. These attributes are only visible to the local application and are not distributed.

You can define custom user attributes so you can associate data with the region or entry and retrieve it later. Unlike the other configuration settings, these attributes are used only by your application.

Note: User attributes are not distributed.

1. Create a Java `Object` with your attribute definitions.
2. Attach the object to the region or to an entry:
    - `Region.setUserAttribute(userAttributeObject)`
    - `Region.getEntry(key).setUserAttribute(userAttributeObject)`
3. Get the attribute value:
    - `Region.getUserAttribute()`
    - `Region.getEntry(key).getUserAttribute()`

This example stores attributes for later retrieval by a cache writer.

```
// Attach a user attribute to a Region with database info for table portfolio
Object myAttribute = "portfolio";
final Region portfolios =
      new RegionFactory().setCacheWriter(new PortfolioDBWriter()).create("Portfolio
s");
Portfolios.setUserAttribute(myAttribute);
```

```
//Implement a cache writer that reads the user attribute setting
public class PortfolioDBWriter extends CacheWriterAdapter {
  public void beforeCreate(RegionEvent event) {
    table = (String)event.getRegion().getUserAttribute();
    // update database table using name from attribute
        . . .
  }
}
```

Limitations and Alternatives

User attributes are not distributed to other processes, so if you need to define each attribute in every process that uses the region or entry. You need to update every instance of the region separately. User attributes are not stored to disk for region persistence or overflow, so they cannot be recovered to reinitialize the region.

If your application requires features not supported by user attributes, an alternative is to create a separate region to hold this data instead. For instance, a region, AttributesRegion, defined by you, could use region names as keys and the user attributes as values. Changes to AttributesRegion would be distributed to other processes, and you could configure the region for persistence or overflow if needed.

Building a New Region with Existing Content

A new region or cluster may need to be loaded with the data of an existing system. There are two approaches to accomplish this task. The approach used depends upon the organization of both the new and the existing cluster.

If both the number and the type of members is the same in both the new and the existing cluster, then the simplest option is to use backup and restore on the persistent disk store contents. Make a full online backup of the persistent data in the disk store of the existing cluster. Copy the files that comprise the backup to the new cluster location. A restore instills the data into the new cluster. See Creating Backups for System Recovery and Operational Management for details on how to make a backup and use the backup to restore a disk store.

Take a different approach when the number or the type of members is *not* the same in both the new and the existing cluster. This approach uses export and import of region data. Export the region data of the existing cluster to create a snapshot. Copy the snapshot to the new cluster location. Import the snapshot into the new cluster. See appropriate sections within Cache and Region Snapshots for details on making and using a snapshot.

Data Entries

The data entry is the key/value pair where you store your data. You can manage your entries individually and in batches. To use domain objects for your entry values and keys, you need to follow VMware GemFire requirements for data storage and distribution.

- **Managing Data Entries**

  Program your applications to create, modify, and manage your cached data entries.

- **Copy on Read Behavior**

  Set the copy-on-read region attribute to cause operations that get data to make a copy of the data, instead of returning a reference to the data.

- **Requirements for Using Custom Classes in Data Caching**

  Follow these guidelines to use custom domain classes for your cached entry keys and values.

Managing Data Entries

Program your applications to create, modify, and manage your cached data entries.

Note: If you do not have the cache's `copy-on-read` attribute set to true, do not change the objects returned from the Java entry access methods. See Copy on Read Behavior.

Keys

VMware GemFire calls `hashCode()` on the key to map an entry within the region. The `hashCode()` return value must be the same for a given key on every server that hosts the region.

An `equals()` call return value on a given key also must be the same on every server that hosts the region.

A key may be a primitive type or a custom class. For custom classes, see Classes Used as Keys.

Do not use an enumerated type (`enum`) for a key. The `enum hashCode()` may not be overridden, and its hash code is based upon an address. Therefore, the return value for a `hashCode()` call can be different on each server, violating the restriction that it must return the same value on every server that hosts the region.

Create and Update Entries

To create or update an entry in the cache, use `Region.put`. For example:

```
String name = ...
String value = ...
this.currRegion.put(name,value);
```

Note: You can also use the `gfsh put` command to add entries to a region, and the `get` command to retrieve entries from a region. See get and put for more information.

If you want only to create the entry (with a null value and with method failure if the entry already exists), use `Region.create` instead.

The getAll Operation

The batch operation `Region.getAll` takes a collection of keys and returns a `Map` of key-value pairs for the provided keys. If a given key does not exist in the region, then that key's value in the returned map will be null.

The putAll Operation

The batch operation `Region.putAll` takes a `Map` of key-value pairs, puts them into the cache, and then distributes them to all other members.

The design of a client application within a client-server design pattern faces the possibility that a partial operation can occur. Some, all, or none of the specified key-value pairs may be written with `putAll`. If either `ServerOperationException` or `ServerConnectivityException` is thrown, it can indicate an incomplete operation.

```
// Retry if the exception may be due to a transient cause.
for (int retry = 0; retry < 3; retry++) {
  throwable = null;
  try {
    region.putAll(map);
  } catch (ServerOperationException e) {
    throwable = e.getCause();
    if (!(e.getCause() instanceof TimeoutException ||
          e.getCause() instanceof LowMemoryException)) {
      // Not a transient exception. Do not retry.
      break;
    }
  } catch (ServerConnectivityException e) {
    throwable = e;
  }
}
```

```
if (throwable != null) {
  // Take appropriate action,
  // such as logging the exception and rethrowing it.
  System.out.println("putAll failed due to " + throwable);
  throw new Exception(throwable);
}
```

Either a `ServerConnectivityException` or a `ServerOperationException` with a cause of `TimeoutException` or `LowMemoryException` can indicate a transient error. A limited quantity of retries of `putAll` may result in a completed operation. A repeated timeout may imply that the `read-timeout` value is not long enough to complete the bulk operation; use the `org.apache.geode.cache.client.PoolFactory.setReadTimeout` method to set the `read-timeout` value.

Client applications that cannot tolerate partial completion of a `putAll` operation may embed the operation into a transaction. See Transactions for details.

The processing of a map with many entries and/or extra-large data values may affect system performance and cause cache update timeouts, especially if the region uses overflow or persistence to disk. The processing may also cause a `LowMemoryException` to be thrown.

The removeAll Operation

The `removeAll` method takes a collection of keys and removes all of the entries for the specified keys from this region. This call performs the equivalent of calling`destroy(Object)` on this region once for each key in the specified collection. If an entry does not exist, then that key is skipped. An `EntryNotFoundException` is not thrown. This operation will be distributed to other caches if the region's scope is not set to `Scope.LOCAL`.

The processing of a map with many entries and/or extra-large data values may affect system performance and cause cache update timeouts, especially if the region uses overflow or persistence to disk. The processing may also cause a `LowMemoryException` to be thrown.

Retrieving Region Entries from Proxy Members

The `Region.values` method call applies to the local region instance only. If you call the `values` method from a client region using the PROXY shortcut, the method call will not be redirected to the server region. To obtain a collection of all values in the Region from a client, you should use interest registration on ALL_KEYS, or use a query.

If you use the `Region.get` method from a proxy member, the method call will redirect to the region on the server if it cannot find the key locally.

Using gfsh to get and put

You can use the gfsh `get` and `put` commands to manage data. See get and put.

For example:

```
get --key=('id':'133abg124') --region=region1

// Retrieving when key type is a wrapper(primitive)/String
```

```
get --key=('133abg124') --region=/region1/region12 --value-class=data.ProfileDetails

get --key=('100L') --region=/region1/region12 --value-class=data.ProfileDetails
--key-class=java.lang.Long
```

```
put --key=('id':'133abg125') --value=('firstname':'James','lastname':'Gosling')
--region=/region1 --key-class=data.ProfileKey --value-class=data.ProfileDetails

put --key=('133abg124') --value=('Hello World!!') --region=/region2

put --key=('100F') --value=('2146547689879658564')  --region=/region1/region12
--key-class=java.lang.Float --value-class=java.lang.Long
```

Copy on Read Behavior

Methods that do a get type of operation receive as a return value a direct reference to the cached object. This provides the value as quickly as possible, but it also makes possible code implementations that could incorrectly modify the referenced object, bypassing the distribution framework and causing region entries that are no longer consistent across cluster members.

The code that has the potential for harming cache consistency by using a reference to access and change a region entry is code that executes within the servers. Examples are cache writers and listeners, transactions, and functions. A client invocation of a get type of operation that is handled by the servers is not subject to this potential for harm, as the clients are in a distinct JVM from the servers, and references do not cross JVM boundaries. That client cannot receive a return value that is a direct reference to a region entry, as the servers hold the region entries and the servers do not reside within the client JVM.

To avoid modification of the referenced object, create a copy in one of two ways:

- Change the entry retrieval behavior for your cache by setting the `copy-on-read` cache attribute to true; its default value is false. When `copy-on-read` is true, all entry access methods return copies of the entries. This protects all server-side code from inadvertently modifying in-place. This attribute will negatively impact performance and memory consumption when a copy is not needed, as it takes time and memory to create the copy. Note that the `copy-on-read` attribute is applied at the cache level; it cannot be set for individual regions.

  There are two ways to set the `copy-on-read` attribute:

  - Set the attribute in the `cache.xml` file that defines the cache.

    ```
    <cache copy-on-read="true">
     ...
    </cache>
    ```

  - Use `gfsh alter runtime` to set the `copy-on-read` attribute once the servers have been started.

- Implement server-side code that creates and uses a copy of the returned object. For objects that are cloneable or serializable, copy the entry value to a new object using `org.apache.geode.CopyHelper.copy`. Example:

```
Object o = region.get(key);
StringBuffer s = (StringBuffer) CopyHelper.copy(o);
// further operations on the region entry value will use s
s.toUpperCase();
```

Always use a `Region` method to then change data in the region. Do not use the reference returned from the entry access method. If the upper case string should become the new value for the region entry:

```
region.put(key, s);
```

Requirements for Using Custom Classes in Data Caching

Follow these guidelines to use custom domain classes for your cached entry keys and values.

CLASSPATH

Each member's `CLASSPATH` must include classes for all objects the member accesses.

- For Java applications, use the standard Java `CLASSPATH`.
- For the cache server process, use the `CLASSPATH` environment variable or the `gfsh start server`'s `--classpath` parameter. See Running VMware GemFire Server Processes.

Data is sent between clients and servers in serialized form and the server stores client data in serialized form. The server does not need to deserialize data to send it to another client or to access it through a `PDXInstance`, but it does need to deserialize it to access it in other ways. The server `CLASSPATH` must include the classes for:

- All entry keys
- Entry values in regions that the server persists to disk
- Entry values the server accesses for any reason other than access using a `PdxInstance` or transfer of the full entry value to a client

For information on `PdxInstance`s, see Data Serialization.

Data Serialization

VMware GemFire serializes data entry keys and values for distribution, so all data that VMware GemFire moves out of the local cache for any reason must be serializable. Additionally, partitioned regions store data in serialized form. Almost every configuration requires serialization.

For information on the requirements and options for data serialization, see Data Serialization.

Classes Used as Keys

The region uses hashing on keys. If you define a custom class to use as a key, for the class, override:

- `equals`

- `hashCode`. The default `hashCode` inherited from `Object` uses identity, which is different in every system member. In partitioned regions, hashing based on identity puts data in the wrong place. For details, see the Java API documentation for `java.lang.Object`.

Do not call `hashCode()` on an `enum` type data member within the key's custom `hashCode()` implementation. The `enum` `hashCode()` may not be overridden, and its hash is based upon an address. Therefore, an enumerated type's `hashCode()` return value can be different on each server, violating the restriction that `hashCode()` must return the same value on every server that hosts the region.

Topologies and Communication

Topologies and Communication explains how to plan and configure VMware GemFire member discovery, peer-to-peer and client/server communication topologies.

- **Topology and Communication General Concepts**

  Before you configure your VMware GemFire members, make sure you understand the options for topology and communication.

- **Peer-to-Peer Configuration**

  Use peer-to-peer configuration to set member discovery and communication within a single cluster.

- **Client/Server Configuration**

  In the client/server architecture, a relatively small server farm manages the cached data of and access to the same data for many client applications. Clients can update and access data efficiently, leaving the servers to manage data distribution to other clients and any synchronization with outside data stores.

- **Multi-site (WAN) Configuration**

  Use the multi-site configuration to scale horizontally between disparate, loosely-coupled clusters. A wide-area network (WAN) is the main use case for the multi-site topology.

Topology and Communication General Concepts

Before you configure your VMware GemFire members, make sure you understand the options for topology and communication.

- **Topology Types**

  The VMware GemFire topology options allow you to scale horizontally and vertically.

- **Planning Topology and Communication**

  Create a topology plan and a detailed list of machines and communication ports that your members will use. Configure your VMware GemFire systems and the communication between systems.

- **How Member Discovery Works**

  VMware GemFire provides various options for member discovery within a cluster and between clients and servers.

- **How Communication Works**

  VMware GemFire uses a combination of TCP and UDP unicast and multicast for communication between members. You can change the default behavior to optimize communication for your system.

- **Using Bind Addresses**

  You use a bind address configuration to send network traffic through non-default network cards and to distribute the load of network traffic for VMware GemFire across multiple cards. If no bind address setting is found, VMware GemFire uses the host machine's default address.

- **Choosing Between IPv4 and IPv6**

  By default, VMware GemFire uses Internet Protocol version 4 for VMware GemFire address specifications. You can switch to Internet Protocol version 6 if all your machines support it. You may lose performance, so you need to understand the costs of making the switch.

Topology Types

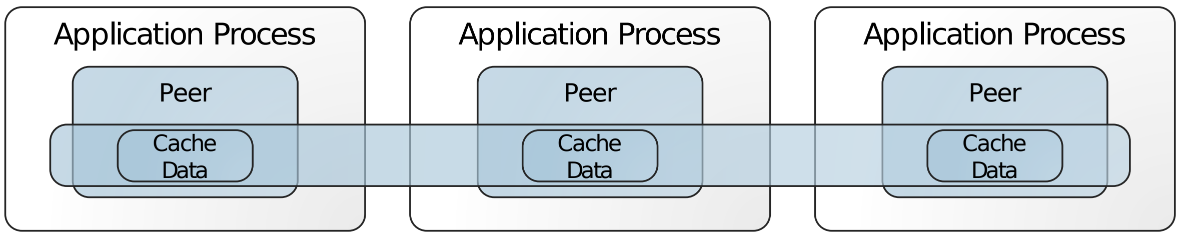
The VMware GemFire topology options allow you to scale horizontally and vertically.

VMware GemFire provides a variety of cache topologies:

- At the core of all systems is the single, peer-to-peer cluster.
- For horizontal and vertical scaling, you can combine individual systems into client/server and multi-site (WAN) topologies:
  - In client/server systems, a small number of server processes manage data and event processing for a much larger client group.
  - In multi-site systems, several geographically disparate systems are loosely coupled into a single, cohesive processing unit.

Peer-to-Peer Configuration

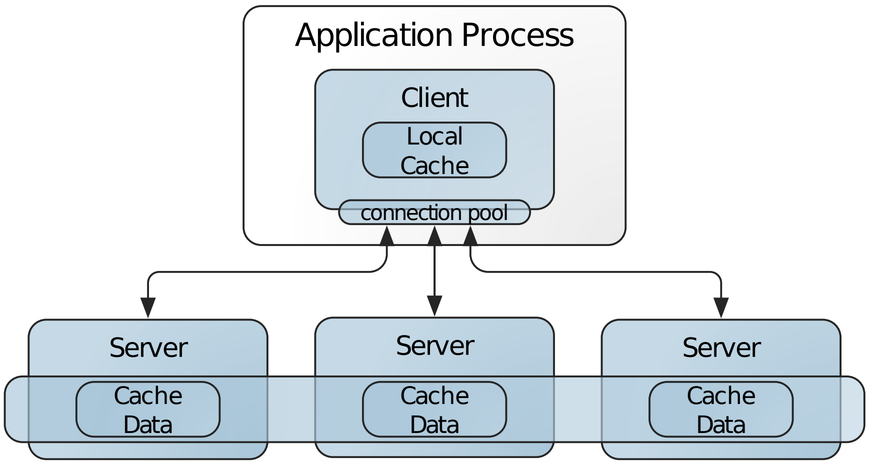
The peer-to-peer cluster is the building block for all VMware GemFire installations. Peer-to-peer alone is the simplest topology. Each cache instance, or member, directly communicates with every other member in the cluster. This cache configuration is primarily designed for applications that need to embed a cache within the application process space and participate in a cluster. A typical example is an application server cluster in which the application and the cache are co-located and share the same heap.



Client/Server Configuration

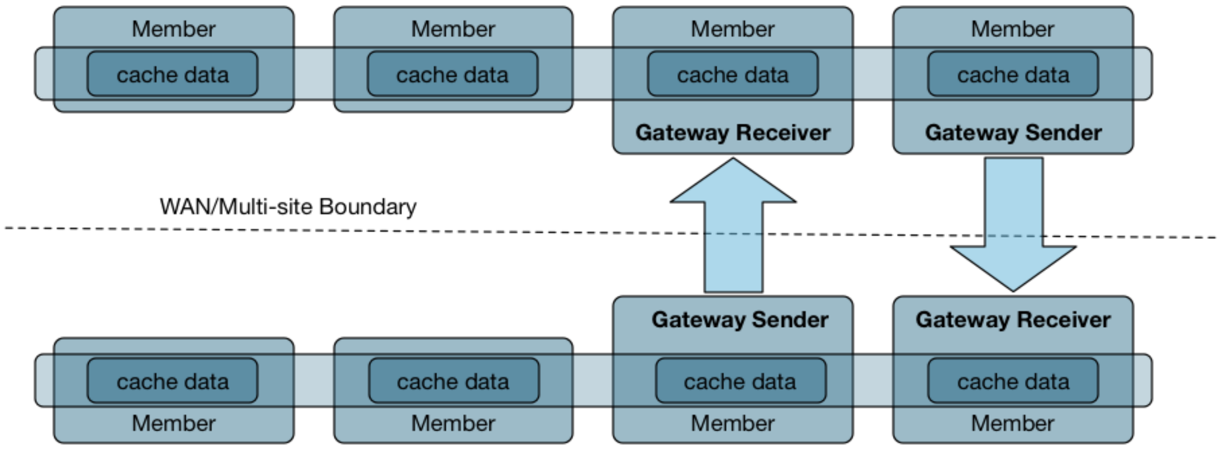
The client/server topology is the model for vertical scaling, where clients typically host a small subset of the data in the application process space and delegate to the server system for the rest.

Compared to peer-to-peer by itself, the client/server architecture provides better data isolation, high fetch performance, and more scalability. If data distribution will put a very heavy load on the network, a client/server architecture usually gives better performance. In any client/server installation, the server system is itself a peer-to-peer system, with data distributed between servers. A client system has a connection pool, which it uses to communicate with servers and other VMware GemFire members. A client may also contain a local cache.



Multi-site Configuration

For horizontal scaling, you can use a loosely coupled multi-site topology. With multi-site, multiple VMware GemFire systems are loosely coupled, generally across geographical distances with slower connections, such as with a WAN. This topology provides better performance than the tight coupling of a single system, and greater independence between locations, so that each site can function on its own if the connection or remote site become unavailable. In a multi-site installation, each individual site is a peer-to-peer or Client/Server system.



Planning Topology and Communication

Create a topology plan and a detailed list of machines and communication ports that your members will use. Configure your VMware GemFire systems and the communication between systems.

Determine Protocols and Addresses

Your configuration governs how your applications find each other and distribute events and data among themselves.

Work with your system administrator to determine the protocols and addresses you will use for membership and communication.

1. For each host machine with more than one network adapter card, decide whether to use the default address or one or more non-default bind addresses. You can use different cards for peer and server.
2. Identify any members you want to run as standalone, isolated members with no member discovery. This can be a good option for clients, because it has faster startup but no peer-to-peer distribution of any kind.
3. For all non-standalone members:
    - Decide how many locators you will use and where they will run. To ensure the most stable startup and availability, use multiple locators on multiple machines.
    - Create a list of your locators' address and port pairs. You will use the list to configure your system members, any clients, and the locators themselves.
    - If you will use multicasting for communication, note the addresses and ports. Select both unique multicast ports and unique addresses for your clusters. **Note:** Use different port numbers for different systems, even if you use different multicast addresses. Some operating systems do not keep communication separate between systems that have unique addresses but the same port number.

Set Up Membership and Communication

Using the protocols and addresses you determined above, do the following:

- Set up membership in your systems.
- Set up communication between system members. See Configuring Peer Communication.
- As needed, set up communication between your systems. See Configuring a Client/Server System.

How Member Discovery Works

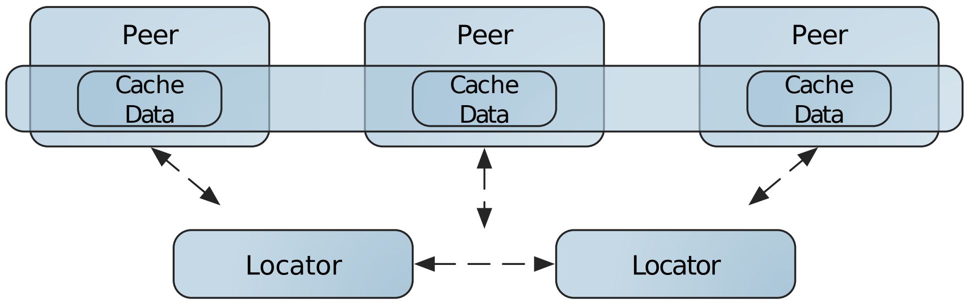
VMware GemFire provides various options for member discovery within a cluster and between clients and servers.

- Peer Member Discovery
- Standalone Member
- Client Discovery of Servers

Peer Member Discovery

Peer member discovery is what defines a cluster. All applications and cache servers that use the same settings for peer discovery are members of the same cluster. Each system member has a unique identity and knows the identities of the other members. A member can belong to only one cluster at a time. Once they have found each other, members communicate directly, independent of the discovery mechanism. In peer discovery, VMware GemFire uses a membership coordinator to manage member joins and departures.

Members discover each other using one or more locators. A locator provides both discovery and load balancing services. Peer locators manage a dynamic list of cluster members. New members connect to one of the locators to retrieve the member list, which it uses to join the system.



Peer Discovery Using Locators

Note: Multiple locators ensure the most stable start up and availability for your cluster.

Standalone Member

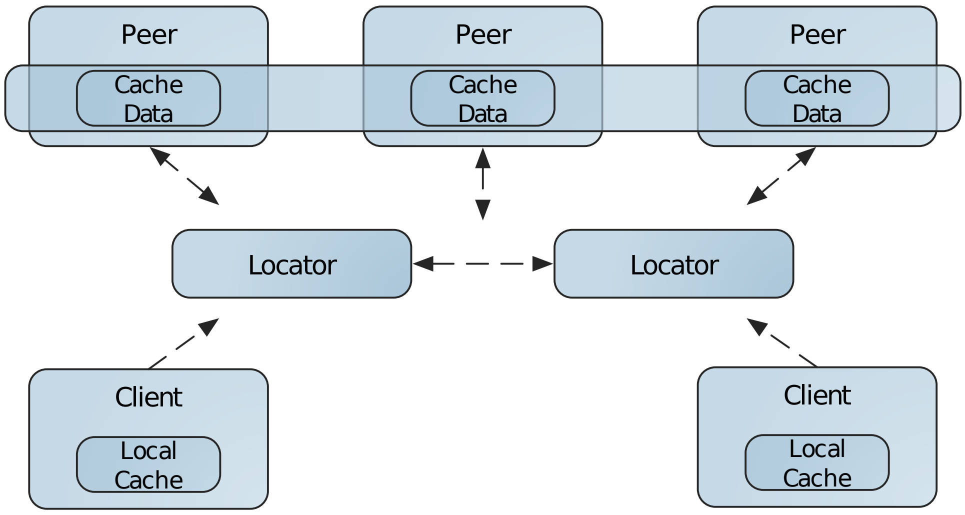
The standalone member has no peers, does no peer discovery, and so does not use locators. It creates a cluster connection only to access the VMware GemFire caching features. Running standalone has a faster startup and is appropriate for any member that is isolated from other applications. The primary use case is for client applications. Standalone members can be accessed and monitored if you enable the member to become a JMX Manager.

Client Discovery of Servers

Locators provide clients with dynamic server discovery and server load balancing. Clients are configured with locator information for the server system, and turn to the locators for directions to the servers to use. The servers can come and go and their capacity to service new client connections can vary. The locators continuously monitor server availability and server load information, providing clients with connection information for the server with the least load at any time.

Note: For performance and cache coherency, clients must run as standalone members or in different clusters than their servers.

You do not need to run any special processes to use locators for server discovery. The locators that provide peer discovery in the server system also provide server discovery for clients to the server system. This is the standard configuration.



Client/Server Discovery Using Locators

Multi-site Discovery

In a multi-site (WAN) configuration, a VMware GemFire cluster uses locators to discover remote VMware GemFire clusters as well as to discover local VMware GemFire members. Each locator in a WAN configuration uniquely identifies the local cluster to which it belongs, and it can also identify locators in remote VMware GemFire clusters to which it will connect for WAN distribution.

When a locator starts up, it contacts each remote locator to exchange information about the available locators and gateway receiver configurations in the remote cluster. In addition to sharing information about its own cluster, a locator shares information that it has obtained from all other connected clusters. Each time a new locator starts up or an existing locator shuts down, the changed information is broadcast to other connected VMware GemFire clusters across the WAN.

See Discovery for Multi-Site Systems for more information.

How Communication Works

VMware GemFire uses a combination of TCP and UDP unicast and multicast for communication between members. You can change the default behavior to optimize communication for your system.

Client/server communication and gateway sender to gateway receiver communication uses TCP/IP sockets. The server listens for client communication at a published address and the client establishes the connection, sending its location. Similarly, the gateway receiver listens for gateway sender communication and the connection is established between sites.

In peer systems, for general messaging and region operations distribution, VMware GemFire uses either TCP or UDP unicast. The default is TCP. You can use TCP or UDP unicast for all communications or you can use it as the default but then can target specific regions to use UDP multicast for operations distribution. The best combination for your installation depends in large part on your data use and event messaging.

TCP

TCP (Transmission Control Protocol) provides reliable in-order delivery of the system messages. TCP is more appropriate than UDP if the data is partitioned, if the cluster is small, or if network loads are unpredictable. TCP is preferable to UDP unicast in smaller clusters because it implements more reliable communications at the operating system level than UDP and its performance can be substantially faster than UDP. As the size of the cluster increases, however, the relatively small overhead of UDP makes it the better choice. TCP adds new threads and sockets to every member, causing more overhead as the system grows.

Note: Even when VMware GemFire is configured to use UDP for messaging, VMware GemFire uses a TCP connection when attempting to detect failed members. See Failure Detection and Membership Views for more details. In addition, the TCP connection's ping is not used for keep alive purposes; it is only used to detect failed members. See TCP/IP KeepAlive Configuration for TCP keep alive configuration.

UDP Unicast and Multicast

UDP (User Datagram Protocol) is a connectionless protocol which uses far fewer resources than TCP. Adding another process to the cluster incurs little overhead for UDP messaging. UDP on its own is not reliable however, and messages are restricted in size to 64k bytes or less, including overhead for message headers. Large messages must be fragmented and transmitted as multiple datagram messages. Consequently, UDP is slower than TCP in many cases and unusable in other cases if network traffic is unpredictable or heavily congested.

UDP is used in VMware GemFire for both unicast and multicast messaging. VMware GemFire implements retransmission protocols to ensure proper delivery of messages over UDP.

UDP Unicast

UDP unicast is the alternative to TCP for general messaging. UDP is more appropriate than TCP for unicast messaging when there are a large number of processes in the cluster, the network is not congested, cached objects are small, and applications can give the cache enough processing time to read from the network. If you disable TCP, VMware GemFire uses UDP for unicast messaging.

For each member, VMware GemFire selects a unique port for UDP unicast communication. You can restrict the range used for the selection by setting `membership-port-range` in the `gemfire.properties` file. Example:

```
membership-port-range=1024-60000
```

Note: In addition to UDP port configuration, the `membership-port-range` property defines the TCP port used for failure detection. See the Reference for a description of the VMware GemFire property.

UDP Multicast

Your options for general messaging and for default region operations messaging is between TCP and UDP unicast. You can choose to replace the default with UDP multicast for operations distribution of some or all of your regions. For every region where you want to use multicast, you set an additional attribute on the region itself.

When multicast is enabled for a region, all processes in the cluster receive all events for the region. Every member receives each message for the region and has to unpack it, schedule it for processing, and then process it, all before discovering whether it is interested in the message. Multicasting is suitable, therefore, for regions that are of general interest in the cluster, where most or all members have the region defined and are interested in receiving most or all messages for the region. Multicasting should not be used for regions that are of little general interest in the cluster.

Multicast is most appropriate when the majority of processes in a cluster are using the same cache regions and need to get updates for them, such as when the processes define replicated regions or have their regions configured to receive all events.

Even if you use multicast for a region, VMware GemFire will send unicast messages when appropriate. If data is partitioned, multicast is not a useful option. Even with multicast enabled, partitioned regions still use unicast for almost all purposes.

Using Bind Addresses

You use a bind address configuration to send network traffic through non-default network cards and to distribute the load of network traffic for VMware GemFire across multiple cards. If no bind address setting is found, VMware GemFire uses the host machine's default address.

Host machines transmit data to the network and receive data from the network through one or more network cards, also referred to as network interface cards (NIC) or LAN cards. A host with more than one card is referred to as a multi-homed host. On multi-homed hosts, one network card is used by default. You can use bind addresses to configure your VMware GemFire members to use non-default network cards on a multi-homed host.

Note: When you specify a non-default card address for a process, all processes that connect to it need to use the same address in their connection settings. For example, if you use bind addresses for your server locators, you must use the same addresses to configure the server pools in your clients.

Use IPv4 or IPv6 numeric address specifications for your bind address settings. For information on these specifications, see Choosing Between IPv4 and IPv6. Do not use host names for your address specifications. Host names resolve to default machine addresses.

Peer and Server Communication

You can configure peer, and server communication so that each communication type uses its own address or types use the same address. If no setting is found for a specific communication type, VMware GemFire uses the host machine's default address.

Note: Bind addresses set through the APIs, like `CacheServer` and `DistributedSystem`, take precedence over the settings discussed here. If your settings are not working, check to make sure there are no bind address settings being done through API calls.

This table lists the settings used for peer and server communication, ordered by precedence. For example, for server communication, VMware GemFire searches first for the cache-server bind address, then the `gfsh start server server-bind-address` setting, and so on until a setting is found or all possibilities are exhausted.

Property Setting Ordered by Precedence	Peer	Server	Gateway Receiver	Syntax
cache.xml <cache-server> bind-address		X		<cache-server>bind-address=*address*
gfsh start server command-line --server-bind-address		X	X	gfsh start server --server-bind-address=*address*
gemfire.properties server-bind-address		X	X	server-bind-address=*address*
gemfire.properties bind-address	X	X	X	bind-address=*address*

For example, a member started with these configurations in its `gemfire.properties` and `cache.xml` files will use two separate addresses for peer and server communication:

```
// gemfire.properties setting for peer communication
bind-address=192.0.2.0
```

```
//cache.xml settings
<cache>

// Server communication
  <cache-server bind-address="192.0.2.1" ...

  <region ...
```

Gateway Receiver Communication

If you are using multi-site (WAN) topology, you can also configure gateway receiver communication (in addition to peer and server communication) so that each communication type uses its own address.

This table lists the settings used for peer, server, and gateway receiver communication, ordered by precedence. For example, for gateway receiver communication, VMware GemFire searches first for a `cache.xml <gateway-receiver> bind-address` setting. If that is not set, VMware GemFire searches for the `gfsh start server server-bind-address` setting, and so on until a setting is found or all possibilities are exhausted.

Property Setting Ordered by Precedence	Peer	Server	Gateway Receiver	Syntax
cache.xml <gateway-receiver> bind-address			X	<gateway-receiver>bind-address=*address*
cache.xml <cache-server> bind-address		X		<cache-server>bind-address=*address*
gfsh start server command-line --server-bind-address		X	X	gfsh start server --server-bind-address=*address*
gemfire.properties server-bind-address		X	X	server-bind-address=*address*
gemfire.properties bind-address	X	X	X	bind-address=*address*

For example, a member started with these configurations in its `gemfire.properties` and `cache.xml` files will use three separate addresses for peer, server, and gateway receiver communication:

```
// gemfire.properties setting for peer communication
bind-address=192.0.2.0
```

```
//cache.xml settings
<cache>
// Gateway receiver configuration
  <gateway-receiver start-port="1530" end-port="1551" bind-address="192.0.2.2"/>

// Server communication
  <cache-server bind-address="192.0.2.1" ...

  <region ...
```

Locator Communication

Set the locator bind address using one of these methods:

- On the gfsh command line, specify the bind address when you start the locator, the same as you specify any other address:

  ```
  gfsh>start locator --name=my_locator --bind-address=ip-address-to-bind --port=p
  ortNumber
  ```

- Inside a VMware GemFire application, take one of the following actions:
  - Automatically start a co-located locator using the gemfire property `start-locator`, and specifying the bind address for it in that property setting.
  - Use `org.apache.geode.distributed.LocatorLauncher` API to start the locator inside your code. Use the `LocatorLauncher.Builder` class to construct an instance of the `LocatorLauncher`, use the `setBindAddress` method to specify the IP address to use and then use the start() method to start a Locator service embedded in your Java application process.

If your locator uses a bind address, make sure every process that accesses the locator has the address as well. For peer-to-peer access to the locator, use the locator's bind address and the locator's port in your `gemfire.properties locators` list. For server discovery in a client/server installation, use the locator's bind address and the locator's port in the locator list you provide to in the client's server pool configuration.

Choosing Between IPv4 and IPv6

By default, VMware GemFire uses Internet Protocol version 4 for VMware GemFire address specifications. You can switch to Internet Protocol version 6 if all your machines support it. You may lose performance, so you need to understand the costs of making the switch.

- IPv4 uses a 32-bit address. IPv4 was the first protocol and is still the main one in use, but its address space is expected to be exhausted within a few years.

- IPv6 uses a 128-bit address. IPv6 succeeds IPv4, and will provide a much greater number of addresses.

Based on current testing with VMware GemFire , IPv4 is generally recommended. IPv6 connections tend to take longer to form and the communication tends to be slower. Not all machines support IPv6 addressing. To use IPv6, all machines in your distributed system must support it or you will have connectivity problems.

Note: Do not mix IPv4 and IPv6 addresses. Use one or the other, across the board.

IPv4 is the default version.

To use IPv6, set the Java property, `java.net.preferIPv6Addresses`, to `true`.

These examples show the formats to use to specify addresses in VMware GemFire .

- IPv4:

  ```
  192.0.2.0
  ```

- IPv6:

  ```
  2001:db8:85a3:0:0:8a2e:370:7334
  ```

Peer-to-Peer Configuration

Use peer-to-peer configuration to set member discovery and communication within a single cluster.

- **Configuring Peer-to-Peer Discovery**

  Peer members discover each other using one or more locators.

- **Configuring Peer Communication**

  By default VMware GemFire uses TCP for communication between members of a single cluster. You can modify this at the member and region levels.

- **Organizing Peers into Logical Member Groups**

  In a peer-to-peer configuration, you can organize members into logical member groups and use those groups to associate specific data or assign tasks to a pre-defined set of members.

Configuring Peer-to-Peer Discovery

Peer members discover each other using one or more locators.

The `gemfire.properties` file can list the locators:

```
locators=<locator1-address>[<port1>],<locator2-address>[<port2>]
```

To run a standalone member, the `gemfire.properties` file disables using locators:

```
locators=
mcast-address=
mcast-port=0
```

Note: Locator settings must be consistent throughout the cluster.

Configuring Peer Communication

By default VMware GemFire uses TCP for communication between members of a single distributed system. You can modify this at the member and region levels.

Before you begin, you should have already determined the address and port settings for multicast, including any bind addresses. See Topology and Communication General Concepts.

See the Reference.

1. **Configure general messaging to use TCP or UDP unicast**.

   TCP is the default protocol for communication. To use it, just make sure you do not have it disabled in `gemfire.properties`. Either have no entry for `disable-tcp`, or have this entry:

   ```
   disable-tcp=false
   ```

   To use UDP unicast for general messaging, add this entry to `gemfire.properties`:

   ```
   disable-tcp=true
   ```

   The `disable-tcp` setting has no effect on the use of TCP locators or the TCP connections used to detect failed members.

2. **Configure any regions you want to distribute using UDP multicast**.

   1. Configure UDP multicast for region messaging, set non-default multicast address and port selections in `gemfire.properties`:

      ```
      mcast-address=<address>
      mcast-port=<port>
      ```

   2. In `cache.xml`, enable multicast for each region that needs multicast messaging:

      ```
      <region-attributes multicast-enabled="true"/>
      ```

      Note: Improperly configured multicast can affect production systems. If you intend to use multicast on a shared network, work with your network administrator and system administrator from the planning stage of the project. In addition, you may need to address interrelated setup and tuning issues at the VMware GemFire, operating system, and network level.

Once your members establish their connections to each other, they will send distributed data and messages according to your configuration.

Organizing Peers into Logical Member Groups

In a peer-to-peer configuration, you can organize members into logical member groups and use those groups to associate specific data or assign tasks to a pre-defined set of members.

You can use logical member groups to deploy JAR applications across multiple members or to execute functions across a member group.

To add a peer to a member group, you can configure the following:

1. Add the member group names to the `gemfire.properties` file for the member. For example:

   ```
   #gemfire.properties
   groups=Portfolios,ManagementGroup1
   ```

   A member can belong to more than one member group. If specifying multiple member groups for a member, use a comma-separated list. Alternatively, if you are using the `gfsh` command interface to start up the member, provide the group name or group names as a parameter.

   For example, to start up a server and associate it with member groups, you could type:

   ```
   gfsh>start server --name=server1 \
   --group=Portfolios,ManagementGroup1
   ```

   For example, to start up a locator and associate it with member groups, you could type:

   ```
   gfsh>start locator --name=locator1 \
   --group=ManagementGroup1
   ```

2. Then you can use the member group names to perform tasks such as deploy applications or execute functions.

   For example, to deploy an application across a member group, you could type the following in `gfsh`:

   ```
   gfsh>deploy --jar=group1_functions.jar --group=ManagementGroup1
   ```

Client/Server Configuration

In the client/server architecture, a relatively small server farm manages the cached data of and access to the same data for many client applications. Clients can update and access data efficiently, leaving the servers to manage data distribution to other clients and any synchronization with outside data stores.

- **Standard Client/Server Deployment**

  In the most common client/server topology, a farm of cache servers provides caching services to many clients. Cache servers have a homogeneous data store in data regions that are replicated or partitioned across the server farm.

- **How Server Discovery Works**

  VMware GemFire locators provide reliable and flexible server discovery services for your clients. You can use all servers for all client requests, or group servers according to function, with the locators directing each client request to the right group of servers.

- **How Client/Server Connections Work**

  The server pools in your VMware GemFire client processes manage all client connection requests to the server tier. To make the best use of the pool functionality, you should understand how the pool manages the server connections.

- **Configuring a Client/Server System**

  Configure your server and client processes and data regions to run your client/server system.

- **Organizing Servers Into Logical Member Groups**

  In a client/server configuration, by putting servers into logical member groups, you can control which servers your clients use and target specific servers for specific data or tasks. You can configure servers to manage different data sets or to direct specific client traffic to a subset of servers, such as those directly connected to a back-end database.

- **Client/Server Example Configurations**

  For easy configuration, you can start with these example client/server configurations and modify for your systems.

- **Fine-Tuning Your Client/Server Configuration**

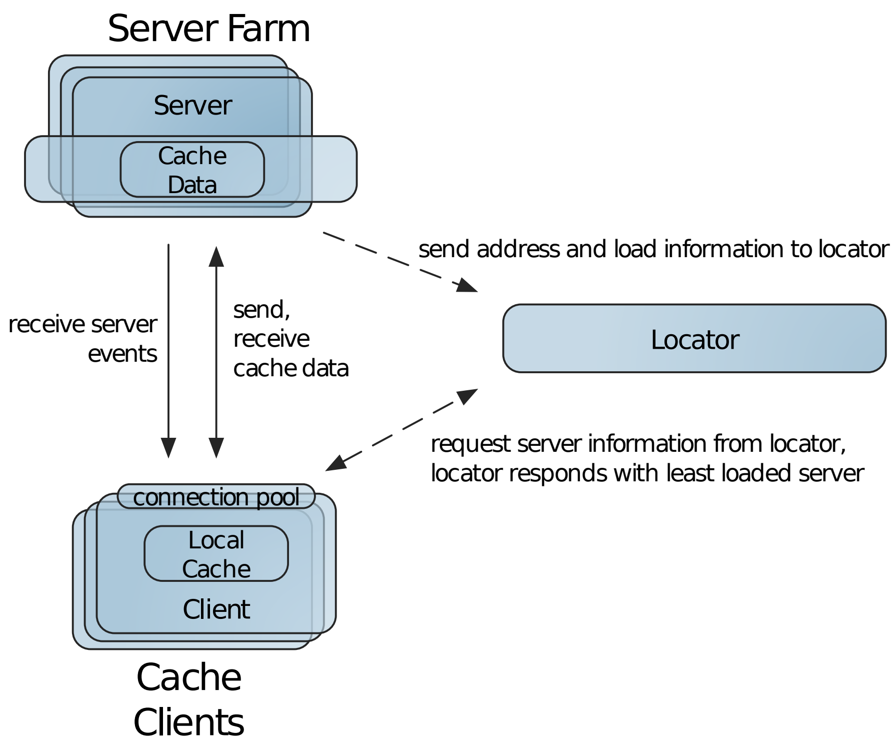
  You can fine-tune your client/server system with server load-balancing. For example, you can configure how often the servers check their load with the cache server `load-poll-interval` property, or configure your own server load metrics by implementing the `org.apache.geode.cache.server` package.

Standard Client/Server Deployment

In the most common client/server topology, a farm of cache servers provides caching services to many clients. Cache servers have a homogeneous data store in data regions that are replicated or partitioned across the server farm.

The client/server data flow proceeds as follows:

- Cache servers send their address and load information to the server locator, if locators are used.
- If locators are used, clients request server connection information from the locator. The locator responds with the address of the least-loaded server.
- The client pool checks its connections periodically for proper server load balancing. The pool rebalances as needed.
- Clients can subscribe to events at startup. Events are streamed automatically from the servers to client listeners and into the client cache.
- Client data updates and data requests that the client cache does not fulfill are forwarded automatically to the servers.


Server Farm
Server
Cache Data
send address and load information to locator
Locator
receive server events
send, receive cache data
request server information from locator, locator responds with least loaded server
connection pool
Local Cache
Client
Cache Clients

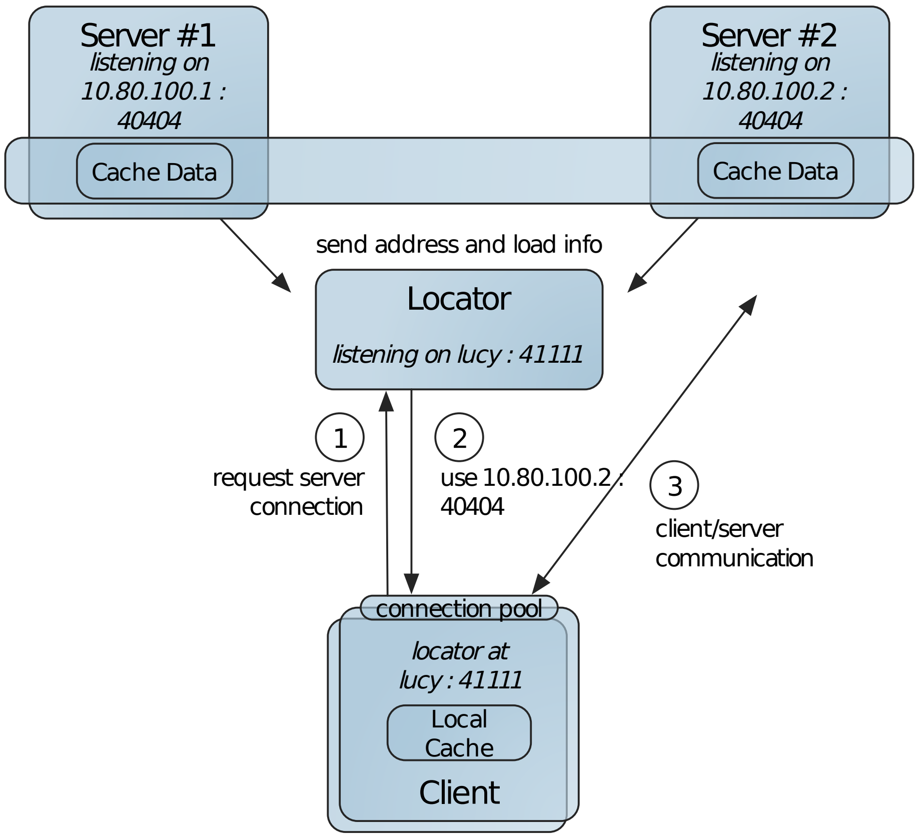
How Server Discovery Works

VMware GemFire locators provide reliable and flexible server discovery services for your clients. You can use all servers for all client requests, or group servers according to function, with the locators directing each client request to the right group of servers.

By default, VMware GemFire clients and servers discover each other on a predefined port (40404) on the localhost. This works, but is not typically the way you would deploy a client/server configuration. The recommended solution is to use one or more dedicated locators. A locator provides both discovery and load balancing services. With server locators, clients are configured with a locator list and locators maintain a dynamic server list. The locator listens at an address and port for connecting clients and gives the clients server information. The clients are configured with locator information and have no configuration specific to the servers.

Basic Configuration

In this figure, only one locator is shown, but the recommended configuration uses multiple locators for high availability.



The locator and servers have the same peer discovery configured in their `gemfire.properties`:

```
locators=lucy[41111]
```

The servers, run on their respective hosts, have this `cache-server` configuration in their `cache.xml`:

```
<cache-server port="40404" ...
```

The client's `cache.xml` `pool` configuration and `region-attributes`:

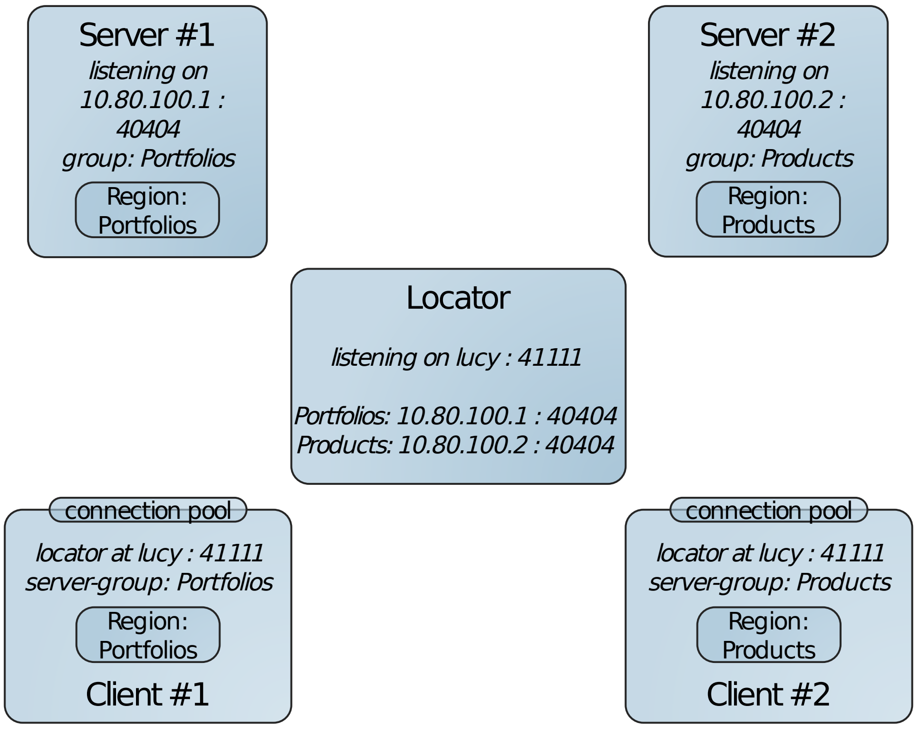
```
<pool name="PoolA" ...
  <locator host="lucy" port="41111">

<region ...
<region-attributes pool-name="PoolA" ...
```

Using Member Groups

You can control which servers are used with named member groups. Do this if you want your servers to manage different data sets or to direct specific client traffic to a subset of servers, such as those directly connected to a back-end database.

To split data management between servers, configure some servers to host one set of data regions and some to host another set. Assign the servers to two separate member groups. Then, define two separate server pools on the client side and assign the pools to the proper corresponding client regions.

In this figure, the client use of the regions is also split, but you could have both pools and both regions defined in all of your clients.



This is the `gemfire.properties` definition for Server 1:

```
#gemfire.properties
groups=Portfolios
```

And the `pool` declaration for Client 1:

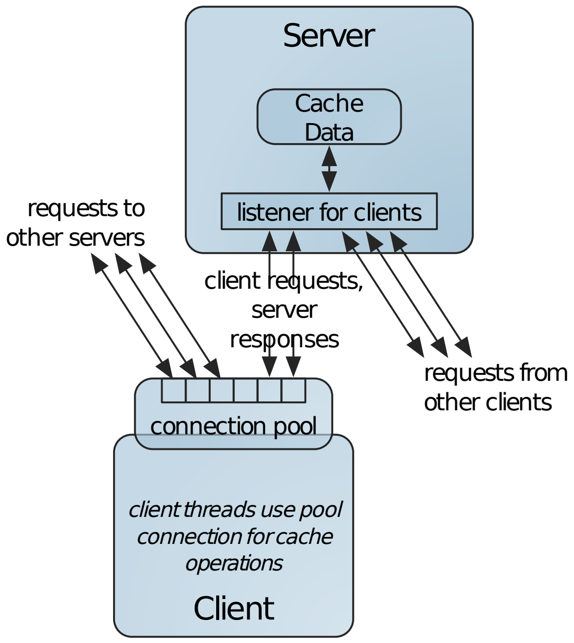
```
<pool name="PortfolioPool" server-group="Portfolios"...
  <locator host="lucy" port="41111">
```

How Client/Server Connections Work

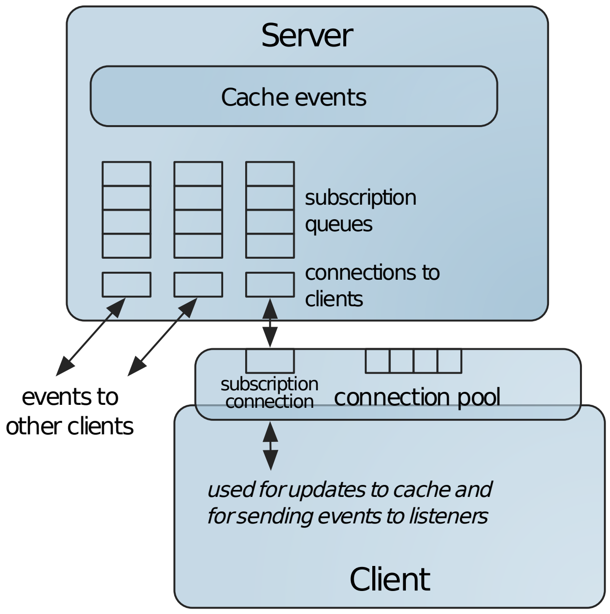
The server pools in your VMware GemFire client processes manage all client connection requests to the server tier. To make the best use of the pool functionality, you should understand how the pool manages the server connections.

Client/server communication is done in two distinct ways. Each kind of communication uses a different type of connection for maximum performance and availability.

- **Pool connections**. The pool connection is used to send individual operations to the server to update cached data, to satisfy a local cache miss, or to run an ad hoc query. Each pool connection goes to a host/port location where a server is listening. The server responds to the request on the same connection. Generally, client threads use a pool connection for an individual operation and then return the connection to the pool for reuse, but you can configure to have connections owned by threads. This figure shows pool connections for one client and one server. At any time, a pool may have from zero to many pool connections to any of the servers.


Server
Cache Data
listener for clients
requests to other servers
client requests, server responses
requests from other clients
connection pool
client threads use pool connection for cache operations
Client

- **Subscription connections**. The subscription connection is used to stream cache events from the server to the client. To use this, set the client attribute `subscription-enabled` to true. The server establishes a queue to asynchronously send subscription events and the pool establishes a subscription connection to handle the incoming messages. The events sent depend on how the client subscribes.



How the Pool Chooses a Server Connection

The pool gets server connection information from the server locators or, alternately, from the static server list.

- **Server Locators**. Server locators maintain information about which servers are available and which has the least load. New connections are sent to the least loaded servers. The pool requests server information from a locator when it needs a new connection. The pool randomly chooses the locator to use and the pool sticks with a locator until the connection fails.
- **Static Server List**. If you use a static server list, the pool shuffles it once at startup, to provide randomness between clients with the same list configuration, and then runs through the list round robin connecting as needed to the next server in the list. There is no load balancing or dynamic server discovery with the static server list.

How the Pool Connects to a Server

When a pool needs a new connection, it goes through these steps until either it successfully establishes a connection, it has exhausted all available servers, or the `free-connection-timeout` is reached.

1. Requests server connection information from the locator or retrieves the next server from the static server list.
2. Sends a connection request to the server.

If the pool fails to connect while creating a subscription connection or provisioning the pool to reach the `min-connections` setting, it logs a fine level message and retries after the time indicated by `ping-interval`.

If an application thread calls an operation that needs a connection and the pool can't create it, the operation returns a `NoAvailableServersException`.

How the Pool Manages Pool Connections

Each `Pool` instance in your client maintains its own connection pool. The pool responds as efficiently as possible to connection loss and requests for new connections, opening new connections as needed. When you use a pool with the server locator, the pool can quickly respond to changes in server availability, adding new servers and disconnecting from unhealthy or dead servers with little or no impact on your client threads. Static server lists require more close attention as the client pool is only able to connect to servers at the locations specified in the list.

The pool adds a new pool connection when one of the following happens:

- The number of open connections is less than the `Pool`'s `min-connections` setting.
- A thread needs a connection, all open connections are in use, and adding another connection would not take the open connection count over the pool's `max-connections` setting. If the max-connections setting has been reached, the thread blocks until a connection becomes available.

The pool closes a pool connection when one of the following occurs:

- The client receives a connectivity exception from the server.

- The server doesn't respond to a direct request or ping within the client's configured `read-timeout` period. In this case, the pool removes all connections to that server.
- The number of pool connections exceeds the pool's `min-connections` setting and the client doesn't send any requests over the connection for the `idle-timeout` period.

When it closes a connection that a thread is using, the pool switches the thread to another server connection, opening a new one if needed.

How the Pool Manages Subscription Connections

The pool's subscription connection is established in the same way as the pool connections, by requesting server information from the locator and then sending a request to the server, or, if you are using a static server list, by connecting to the next server in the list.

The server sends ping messages once per second by a task scheduled in a timer. You can adjust the interval with the system property `gemfire.serverToClientPingPeriod`, specified in milliseconds. The server sends its ping-interval setting to the client. The client then uses this and a multiplier to establish a read-timeout in the cache.

You can set the client property `subscription-timeout-multiplier` to enable timeout of the subscription feed with failover to another server.

Value options include:

- A value of zero (the default) deactivates timeouts.
- A value of one or more times out the server connection after the specified number of ping intervals have elapsed. A value of one is not recommended.

How the Pool Conditions Server Load

When locators are used, the pool periodically conditions its pool connections. Each connection has an internal lifetime counter. When the counter reaches the configured `load-conditioning-interval`, the pool checks with the locator to see if the connection is using the least loaded server. If not, the pool establishes a new connection to the least loaded server, silently puts it in place of the old connection, and closes the old connection. In either case, when the operation completes, the counter starts at zero. Conditioning happens behind the scenes and does not affect your application's connection use. This automatic conditioning allows very efficient upscaling of your server pool. It is also useful following planned and unplanned server outages, during which time the entire client load will have been placed on a subset of the normal set of servers.

Configuring a Client/Server System

Configure your server and client processes and data regions to run your client/server system.

Prerequisites

- Configure your server system using locators for member discovery. See Configuring Peer-to-Peer Discovery and Managing a Peer or Server Cache.
- Configure your clients as standalone applications. See Managing a Client Cache.
- Be familiar with cache region configuration. See Data Regions.

- Be familiar with server and client configuration properties. See cache.xml.

Procedure

1. Configure servers to listen for clients by completing one or both of the following tasks.
    - Configure each application server as a server by specifying the `<cache-server>` element in the application's `cache.xml` and optionally specifying a non-default port to listen on for client connections.

      For example:

      ```
      <cache-server port="40404" ... />
      ```
    - Optional. Configure each `cacheserver` process with a non-default port to listen on for client connections.

      For example:

      ```
      prompt> cacheserver start -port="44454"
      ```
2. Configure clients to connect to servers. In the client `cache.xml`, use the server system's locator list to configure your client server pools and configure your client regions to use the pools. For example:

   ```
   <client-cache>
      <pool name="publisher" subscription-enabled="true">
         <locator host="lucy" port="41111"/>
         <locator host="lucy" port="41111"/>
      </pool>
      ...
      <region name="clientRegion" ...
         <region-attributes pool-name="publisher" ...
   ```

   You do not need to provide the complete list of locators to the clients at startup, but you should provide as complete a list as possible. The locators maintain a dynamic list of locators and servers and provide the information to the clients as needed.
3. Configure the server data regions for client/server work, following these guidelines. These do not need to be performed in this order. 1.

   ```
   Configure your server regions as partitioned or replicated, to provide a cohere
   nt cache view of server data to all clients.
   **Note:**
   If you do not configure your server regions as partitioned or replicated, you c
   an get unexpected results with calls that check server region contents, such as
   `keySetOnServer` and `containsKeyOnServer`. You might get only partial results,
   and you might also get inconsistent responses from two consecutive calls. These
   results occur because the servers report only on their local cache content and,
   without partitioned or replicated regions, they might not have a complete view
   of the data in their local caches.
   ```

    1. When you define your replicated server regions, use any of the `REPLICATE RegionShortcut` settings except for `REPLICATE_PROXY`. Replicated server regions must have `distributed-ack` or `global scope`, and every server that defines the

region must store data. The region shortcuts use `distributed-ack` scope and all non-proxy settings store data. 3.

When you define your partitioned server regions, use the `PARTITION RegionShortcut` options. You can have local data storage in some servers and no local storage in others.

When you start the server and client systems, the client regions will use the server regions for cache misses, event subscriptions, querying, and other caching activities.

What to do next

Configure your clients to use the cache and to subscribe to events from the servers as needed by your application. See Configuring Client/Server Event Messaging.

Organizing Servers Into Logical Member Groups

In a client/server configuration, by putting servers into logical member groups, you can control which servers your clients use and target specific servers for specific data or tasks. You can configure servers to manage different data sets or to direct specific client traffic to a subset of servers, such as those directly connected to a back-end database.

You can also define member groups to deploy JARs in parallel or to perform administrative commands across a member group.

To add servers to a member group, you can configure the following:

1. Add the member group names to the `gemfire.properties` file for the server. For example:

   ```
   groups=Portfolios,ManagementGroup1
   ```

   A server can belong to more than one member group. If specifying multiple group membership for the server, use a comma-separated list. Alternatively, if you are using the `gfsh` command interface to start up the server, provide a group name as a parameter:

   ```
   gfsh>start server --name=server1 \
   --group=Portfolios,ManagementGroup1
   ```

2. To configure a client to connect to a specific member group, modify the client's `cache.xml` file to define a distinct pool for each `server-group` and assign the pools to the corresponding client regions:

   ```
   <pool name="PortfolioPool" server-group="Portfolios" ...
     <locator host="lucy" port="41111">
     ...
   </pool>
      ...
   <region name="clientRegion" ...
     <region-attributes pool-name="PortfolioPool" ...
   ```

Client/Server Example Configurations

For easy configuration, you can start with these example client/server configurations and modify for your systems.

Examples of Standard Client/Server Configuration

Generally, locators and servers use the same properties file, which lists locators as the discovery mechanism for peer members and for connecting clients. For example:

```
mcast-port=0
locators=localhost[41111]
```

On the machine where you wish to run the locator (in this example, 'localhost'), you can start the locator from a gfsh prompt:

```
gfsh>start locator --name=locator_name --port=41111
```

Or directly from a command line:

```
prompt# gfsh start locator --name=locator_name --port=41111
```

Specify a name for the locator that you wish to start on the localhost. If you do not specify the member name, gfsh will automatically pick a random name. This is useful for automation.

The server's `cache.xml` declares a `cache-server` element, which identifies the JVM as a server in the cluster.

```
<cache>
  <cache-server port="40404" ... />
  <region . . .
```

Once the locator and server are started, the locator tracks the server as a peer in its cluster and as a server listening for client connections at port 40404.

You can also configure a cache server using the `gfsh` command-line utility. For example:

```
gfsh>start server --name=server1 --server-port=40404
```

See `start server`.

The client's `cache.xml` `<client-cache>` declaration automatically configures it as a standalone VMware GemFire application.

The client's `cache.xml`:

- Declares a single connection pool with the locator as the reference for obtaining server connection information.
- Creates `cs_region` with the client region shortcut configuration, `CACHING_PROXY`. This configures it as a client region that stores data in the client cache.

There is only one pool defined for the client, so the pool is automatically assigned to all client regions.

```
<client-cache>
    <pool name="publisher" subscription-enabled="true">
       <locator host="localhost" port="41111"/>
    </pool>
    <region name="cs_region" refid="CACHING_PROXY">
    </region>
</client-cache>
```

With this, the client is configured to go to the locator for the server connection location. Then any cache miss or put in the client region is automatically forwarded to the server.

Example—Standalone Publisher Client, Client Pool, and Region

The following API example walks through the creation of a standalone publisher client and the client pool and region.

```
public static ClientCacheFactory connectStandalone(String name) {
    return new ClientCacheFactory()
        .set("log-file", name + ".log")
        .set("statistic-archive-file", name + ".gfs")
        .set("statistic-sampling-enabled", "true")
        .set("cache-xml-file", "")
        .addPoolLocator("localhost", LOCATOR_PORT);
}

private static void runPublisher() {
    ClientCacheFactory ccf = connectStandalone("publisher");
    ClientCache cache = ccf.create();
    ClientRegionFactory<String,String> regionFactory =
        cache.createClientRegionFactory(PROXY);
    Region<String, String> region = regionFactory.create("DATA");

    //... do work ...

    cache.close();
}
```

Example—Standalone Subscriber Client

This API example creates a standalone subscriber client using the same `connectStandalone` method as the previous example.

```
private static void runSubscriber() throws InterruptedException {
    ClientCacheFactory ccf = connectStandalone("subscriber");
    ccf.setPoolSubscriptionEnabled(true);
    ClientCache cache = ccf.create();
    ClientRegionFactory<String,String> regionFactory =
        cache.createClientRegionFactory(PROXY);
    Region<String, String> region = regionFactory
        .addCacheListener(new SubscriberListener())
        .create("DATA");
    region.registerInterestRegex(".*", // everything
        InterestResultPolicy.NONE,
        false/*isDurable*/);
    SubscriberListener myListener =
        (SubscriberListener)region.getAttributes().getCacheListeners()[0];
    System.out.println("waiting for publisher to do " + NUM_PUTS + " puts...");
    myListener.waitForPuts(NUM_PUTS);
    System.out.println("done waiting for publisher.");
    cache.close();
}
```

Example of a Static Server List in Client/Server Configuration

You can specify a static server list instead of a locator list in the client configuration. With this configuration, the client's server information does not change for the life of the client member. You do not get dynamic server discovery, server load conditioning, or the option of logical server grouping. This model is useful for very small deployments, such as test systems, where your server pool is stable. It avoids the administrative overhead of running locators.

This model is also suitable if you must use hardware load balancers. You can put the addresses of the load balancers in your server list and allow the balancers to redirect your client connections.

The client's server specification must match the addresses where the servers are listening. In the server cache configuration file, here are the pertinent settings.

```
<cache>
    <cache-server port="40404" ... />
      <region . . .
```

The client's `cache.xml` file declares a connection pool with the server explicitly listed and names the pool in the attributes for the client region. This XML file uses a region attributes template to initialize the region attributes configuration.

```
<client-cache>
    <pool name="publisher" subscription-enabled="true">
        <server host="localhost" port="40404"/>
    </pool>
    <region name="cs_region" refid="CACHING_PROXY">
    </region>
</client-cache>
```

Fine-Tuning Your Client/Server Configuration

You can fine-tune your client/server system with server load-balancing. For example, you can configure how often the servers check their load with the cache server `load-poll-interval` property, or configure your own server load metrics by implementing the `org.apache.geode.cache.server` package.

How Server Load Conditioning Works

When the client pool requests connection information from the server locator, the locator returns the least-loaded server for the connection type. The pool uses this "best server" response to open new connections and to condition (rebalance) its existing pool connections.

- The locator tracks server availability and load according to information that the servers provide. Each server probes its load metrics periodically and, when it detects a change, sends new information to the locator. This information consists of current load levels and estimates of how much load would be added for each additional connection. The locator compares the load information from its servers to determine which servers can best handle more connections.

- You can configure how often the servers check their load with the cache server's `load-poll-interval`. You might want to set it lower if you find your server loads fluctuating too much during normal operation. The lower you set it, however, the more overhead your load balancing will use.
- Between updates from the servers, the locators estimate which server is the least loaded by using the server estimates for the cost of additional connections. For example, if the current pool connection load for a server's connections is 0.4 and each additional connection would add 0.1 to its load, the locator can estimate that adding two new pool connections will take the server's pool connection load to 0.6.
- Locators do not share connection information among themselves. These estimates provide rough guidance to the individual locators for the periods between updates from the servers.

VMware GemFire provides a default utility that probes the server and its resource usage to give load information to the locators. The default probe returns the following load metrics: - The pool connection load is the number of connections to the server divided by the server's `max-connections` setting. This means that servers with a lower `max-connections` setting receives fewer connections than servers with a higher setting. The load is a number between 0 and 1, where 0 means there are no connections, and 1 means the server is at `max-connections`. The load estimate for each additional pool connection is 1/`max-connections`. - The subscription connection load is the number of subscription queues hosted by this server. The load estimate for each additional subscription connection is 1.

To use your own server load metrics instead of the default, implement the `ServerLoadProbe` or `ServerLoadProbeAdapter` and related interfaces and classes in the `org.apache.geode.cache.server` package. The load for each server is weighed relative to the loads reported by other servers in the system.

Multi-site (WAN) Configuration

Use the multi-site configuration to scale horizontally between disparate, loosely-coupled clusters. A wide-area network (WAN) is the main use case for the multi-site topology.

- **How Multi-site (WAN) Systems Work**

  The VMware GemFire multi-site implementation connects disparate clusters. The systems act as one when they are coupled, and they act as independent systems when communication between sites fails. The coupling is tolerant of weak or slow links between cluster sites. A wide-area network (WAN) is the main use case for the multi-site topology.

- **Multi-site (WAN) Topologies**

  To configure your multi-site topology, you should understand the recommended topologies and the topologies to avoid.

- **Configuring a Multi-site (WAN) System**

  Plan and configure your multi-site topology, and configure the regions that will be shared between systems.

- **Filtering Events for Multi-Site (WAN) Distribution**

  You can optionally create gateway sender and/or gateway receiver filters to control which events are queued and distributed to a remote site, or to modify the data stream that is transmitted between VMware GemFire sites.

- **Resolving Conflicting Events**

  You can optionally create a `GatewayConflictResolver` cache plug-in to decide whether a potentially conflicting event that was delivered from another site should be applied to the local cache.

How Multi-site (WAN) Systems Work

The VMware GemFire multi-site implementation connects distinct clusters. The clusters act as one distributed system when they are coupled, and they act as independent systems when communication between sites fails. The coupling is tolerant of weak or slow links between cluster sites. A wide-area network (WAN) is the main use case for the multi-site topology.

- **Overview of Multi-site Caching**

  A multi-site installation consists of two or more clusters that are loosely coupled. Each site manages its own cluster, but region data is distributed to remote sites using one or more logical connections.

- **Consistency for WAN Updates**

  VMware GemFire ensures that all copies of a region eventually reach a consistent state on all members and clients that host the region, including VMware GemFire members that distribute region events across a WAN.

- **Discovery for Multi-Site Systems**

  Each VMware GemFire cluster in a WAN configuration uses locators to discover remote clusters as well as local members.

- **Gateway Senders**

  A VMware GemFire cluster uses a *gateway sender* to distribute region events to another, remote VMware GemFire cluster. You can create multiple gateway sender configurations to distribute region events to multiple remote clusters, and/or to distribute region events concurrently to another remote cluster.

- **Gateway Receivers**

  A gateway receiver configures a physical connection for receiving region events from gateway senders in one or more remote VMware GemFire clusters.

Multi-site (WAN) Topologies

To configure your multi-site topology, you should understand the recommended topologies and the topologies to avoid.

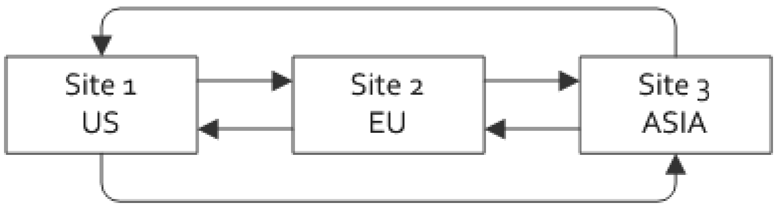
This section describes VMware GemFire's support for various topologies. Depending on your application needs, there may be several topologies that work. These are considerations to keep in mind:

- When a VMware GemFire site receives a message from a gateway sender, it forwards it to the other sites it knows about, excluding those sites that it knows have already seen the message. Each message contains the initial sender's ID and the ID of each of the sites the initial sender sent to, so no site forwards to those sites. However, messages do not pick up the ID of the sites they pass through, so it is possible in certain topologies for more than one copy of a message to be sent to one site.
- In some configurations, the loss of one site affects how other sites communicate with one another.

Fully Connected Mesh Topology

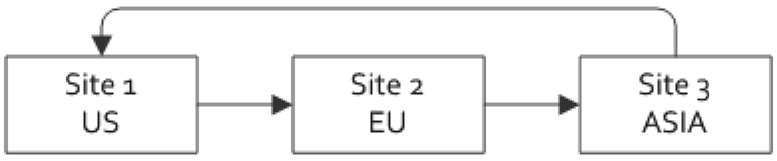
A fully connected mesh network topology is one in which all sites know about each other. This is a robust configuration, as any one of the sites can go down without disrupting communication between the other sites. A fully connected mesh topology also guarantees that no site receives multiple copies of the same message.

A fully connected mesh with three sites is shown in this figure. In this scenario, if site 1 sends an update to site 2, site 2 forwards to site 3. If site 1 sends an update to sites 2 and 3, neither forwards to the other. This is likewise true for any other initiating site. If any site is removed, the remaining two are still fully connected.



Ring Topology

A ring topology is one in which each site forwards information to one other site, and the sites are connected in a circular manner. This figure shows a ring with three sites. In this topology, if site 1 sends updates to site 2, site 2 forwards the updates to site 3. No updates are forwarded to the original sender, so site 3 does not send the updates back to site 1.

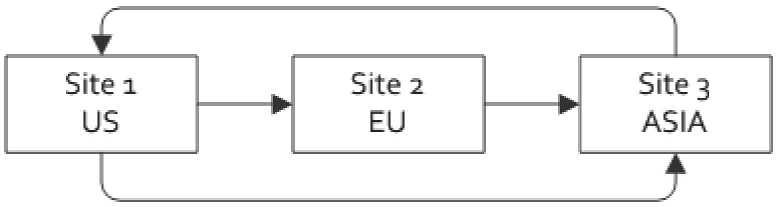


A ring topology guarantees that every site receives one copy of each message sent by any site. In a ring, every site must stay up to maintain the connection. The failure of any site breaks the ability for updates to reach all sites. If site 2 went down, for example, site 3 could send to site 1, but site 1 could not send to site 3.

Hybrid Multi-site Topology

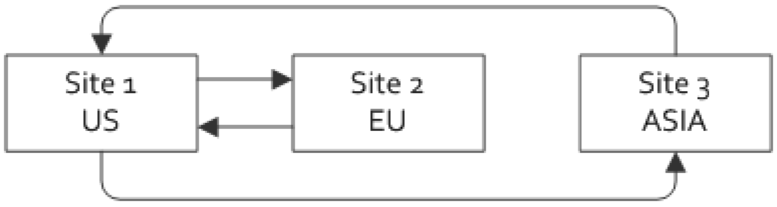
There are numerous hybrid network topologies. Some of the sites are fully connected, while others form a ring.

The following figure shows a hybrid topology that forms a ring, with an extra connection that fully connects sites 1 and 3.



With this hybrid topology, if site 2 went down, it would not affect communication between sites 1 and 3. If site 3 went down, however, site 2 would not be able to send to site 1.

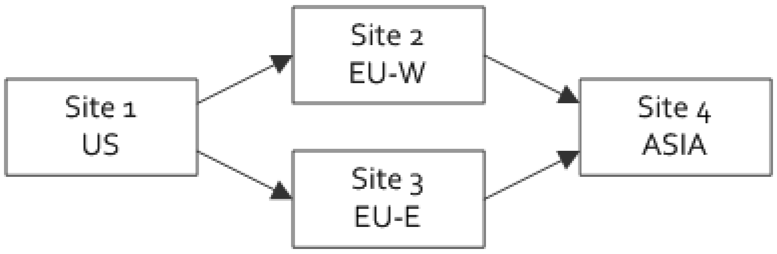
A second example hybrid topology is shown in the figure below. In this tree topology with site 1 as the root of the tree, sites 2 and 3 do not communicate with each another. This topology works for an application in which site 1 is a producer and the consumers (sites 2 and 3) have nothing to gain from being connected to each other. This topology also guarantees that no site receives the same update twice.



Unsupported Topologies

Topologies in which the same update may be delivered twice to a particular site do not work and are unsupported.

The DAG topology shown in this figure is an example of an unsupported technology. Site 4 will receive more than one copy of the same message when site 1 sends a message to sites 2 and 3, and sites 2 and 3 each forward the message to site 4.



Configuring a Multi-site (WAN) System

Plan and configure your multi-site topology, and configure the regions that will be shared between systems.

Prerequisites

Before you start, you should understand how to configure membership and communication in peer-to-peer systems using locators. See Configuring Peer-to-Peer Discovery and Configuring Peer Communication.

WAN deployments increase the messaging demands on a VMware GemFire system. To avoid hangs related to WAN messaging, always use the default setting of `conserve-sockets=false` for VMware GemFire members that participate in a WAN deployment. See Configuring Sockets in Multi-Site (WAN) Deployments and Making Sure You Have Enough Sockets.

Main Steps

Use the following steps to configure a multi-site system:

1. Plan the topology of your multi-site system. See Multi-site (WAN) Topologies for a description of different multi-site topologies.
2. Configure membership and communication for each cluster in your multi-site system. You must use locators for peer discovery in a WAN configuration. See Configuring Peer-to-Peer Discovery. Start each cluster using a unique `distributed-system-id` and identify remote clusters using `remote-locators`. For example:

   ```
   mcast-port=0
   locators=<locator1-address>[<port1>],<locator2-address>[<port2>]
   distributed-system-id=1
   remote-locators=<remote-locator-addr1>[<port1>],<remote-locator-addr2>[<port2>]
   ```

3. Configure the gateway senders that you will use to distribute region events to remote systems. See Configure Gateway Senders.
4. Create the data regions that you want to participate in the multi-site system, specifying the gateway sender(s) that each region should use for WAN distribution. Configure the same regions in the target clusters to apply the distributed events. See Create Data Regions for Multi-site Communication.
5. Configure gateway receivers in each VMware GemFire cluster that will receive region events from another cluster. See Configure Gateway Receivers.
6. Start cluster member processes in the correct order (locators first, followed by data nodes) to ensure efficient discovery of WAN resources. See Starting Up and Shutting Down Your System.
7. (Optional.) Deploy custom conflict resolvers to handle resolve potential conflicts that are detected when applying events from over a WAN. See Resolving Conflicting Events.
8. (Optional.) Deploy WAN filters to determine which events are distributed over the WAN, or to modify events as they are distributed over the WAN. See Filtering Events for Multi-Site (WAN) Distribution.
9. (Optional.) Configure persistence, conflation, and/or dispatcher threads for gateway sender queues using the instructions in Configuring Multi-Site (WAN) Event Queues.

Configure Gateway Senders

Each gateway sender configuration includes:

- A unique ID for the gateway sender configuration.
- The distributed system ID of the remote site to which the sender propagates region events.

- A property that specifies whether the gateway sender is a serial gateway sender or a parallel gateway sender.
- Optional properties that configure the gateway sender queue. These queue properties determine features such the amount of memory used by the queue, whether the queue is persisted to disk, and how one or more gateway sender threads dispatch events from the queue.

Note: To configure a gateway sender that uses gfsh to create the cache.xml configurations described below, as well as other options, see create gateway-sender.

See WAN Configuration for more information about individual configuration properties.

1. For each VMware GemFire system, choose the members that will host a gateway sender configuration and distribute region events to remote sites:
    - You must deploy a parallel gateway sender configuration on each VMware GemFire member that hosts a region that uses the sender. Regions using the same parallel gateway sender ID must be colocated.
    - You may choose to deploy a serial gateway sender configuration on one or more VMware GemFire members in order to provide high availability. However, only one instance of a given serial gateway sender configuration distributes region events at any given time.
2. Configure each gateway sender on a VMware GemFire member using gfsh, `cache.xml` or Java API:
    - **gfsh configuration command**

      ```
      gfsh>create gateway-sender --id="sender2" --parallel=true --remote-distri
      buted-system-id="2"

      gfsh>create gateway-sender --id="sender3" --parallel=true --remote-distri
      buted-system-id="3"
      ```
    - **cache.xml configuration**

      These example `cache.xml` entries configure two parallel gateway senders to distribute region events to two remote VMware GemFire clusters (clusters “2” and “3”):

      ```
      <cache>
        <gateway-sender id="sender2" parallel="true"
         remote-distributed-system-id="2"/>
        <gateway-sender id="sender3" parallel="true"
         remote-distributed-system-id="3"/>
         ...
      </cache>
      ```
    - **Java configuration**

      This example code shows how to configure a parallel gateway sender using the API:

      ```
      // Create or obtain the cache
      Cache cache = new CacheFactory().create();
      ```

```
// Configure and create the gateway sender
GatewaySenderFactory gateway = cache.createGatewaySenderFactory();
gateway.setParallel(true);
GatewaySender sender = gateway.create("sender2", "2");
sender.start();
```

3. Depending on your applications, you may need to configure additional features in each gateway sender. Things you need to consider are:

   - The maximum amount of memory each gateway sender queue can use. When the queue exceeds the configured amount of memory, the contents of the queue overflow to disk. For example:

     ```
     gfsh>create gateway-sender --id=sender2 --parallel=true --remote-distribu
     ted-system-id=2 --maximum-queue-memory=150
     ```

     In cache.xml:

     ```
     <gateway-sender id="sender2" parallel="true"
      remote-distributed-system-id="2"
      maximum-queue-memory="150"/>
     ```

   - Whether to enable disk persistence, and whether to use a named disk store for persistence or for overflowing queue events. See Persisting an Event Queue. For example:

     ```
     gfsh>create gateway-sender --id=sender2 --parallel=true --remote-distribu
     ted-system-id=2 \
     --maximum-queue-memory=150 --enable-persistence=true --disk-store-name=cl
     uster2Store
     ```

     In cache.xml:

     ```
     <gateway-sender id="sender2" parallel="true"
      remote-distributed-system-id="2"
      enable-persistence="true" disk-store-name="cluster2Store"
      maximum-queue-memory="150"/>
     ```

   - The number of dispatcher threads to use for processing events from each each gateway queue. The `dispatcher-threads` attribute of the gateway sender specifies the number of threads that process the queue (default of 5). For example:

     ```
     gfsh>create gateway-sender --id=sender2 --parallel=true --remote-distribu
     ted-system-id=2 \
     --dispatcher-threads=2 --order-policy=partition
     ```

     In cache.xml:

     ```
     <gateway-sender id="sender2" parallel="false"
      remote-distributed-system-id="2"
      dispatcher-threads="2" order-policy="partition"/>
     ```

     Note: When multiple dispatcher threads are configured for a serial queue, each thread operates on its own copy of the gateway sender queue. Queue configuration attributes such as `maximum-queue-memory` are repeated for each dispatcher thread that you configure.

See Configuring Dispatcher Threads and Order Policy for Event Distribution.

- For serial gateway senders (parallel=false) that use multiple `dispatcher-threads`, also configure the ordering policy to use for dispatching the events. See Configuring Dispatcher Threads and Order Policy for Event Distribution.
- Determine whether you should conflate events in the queue. See Conflating Events in a Queue.

Note: The gateway sender configuration for a specific sender `id` must be identical on each VMware GemFire member that hosts the gateway sender.

Create Data Regions for Multi-site Communication

When using a multi-site configuration, you choose which data regions to share between sites. Because of the high cost of distributing data between disparate geographical locations, not all changes are passed between sites.

Note these important restrictions on the regions:

- Replicated regions cannot use a parallel gateway sender. Use a serial gateway sender instead.
- In addition to configuring regions with gateway senders to distribute events, you must configure the same regions in the target clusters to apply the distributed events. The region name in the receiving cluster must exactly match the region name in the sending cluster.
- Regions using the same parallel gateway sender ID must be colocated.
- If any gateway sender configured for the region has the `group-transaction-events` flag set to true, then the regions involved in transactions must all have the same gateway senders configured with this flag set to true. This requires careful configuration of regions with gateway senders according to the transactions expected in the system.

  Example: Assuming the following scenario:

  - Gateway-senders:
    - sender1: group-transaction-events=true
    - sender2: group-transaction-events=true
    - sender3: group-transaction-events=true
    - sender4: group-transaction-events=false
  - Regions:
    - region1: gateway-sender-ids=sender1,sender2,sender4
      type: partition
      colocated-with: region2,region3
    - region2: gateway-sender-ids=sender1,sender2
      type: partition
      colocated with: region1,region3

        - region3: gateway-sender-ids=sender3
          type: partition
          colocated with: region1,region2
        - region4: gateway-sender-ids=sender4
          type: replicated
    - Events for the same transaction will be guaranteed to be sent in the same batch depending on the events involved in the transaction:
        - For transactions containing events for region1 and region2, it will be guaranteed that events for those transactions will be delivered in the same batch by sender1 and sender2.
        - For transactions containing events for region1, region2 and region3, it will NOT be guaranteed that events for those transactions will be delivered in the same batch .
        - For transactions containing events for region3, it will be guaranteed that events for those transactions will be delivered in the same batch.
        - For transactions containing events for region4, it will NOT be guaranteed that events for those transactions will be delivered in the same batch.

After you define gateway senders, configure regions to use the gateway senders to distribute region events.

- **gfsh Configuration**

  ```
  gfsh>create region --name=customer --gateway-sender-id=sender2,sender3
  ```

  or to modify an existing region:

  ```
  gfsh>alter region --name=customer --gateway-sender-id=sender2,sender3
  ```

- **cache.xml Configuration**

  Use the `gateway-sender-ids` region attribute to add gateway senders to a region. To assign multiple gateway senders, use a comma-separated list. For example:

  ```
  <region-attributes gateway-sender-ids="sender2,sender3">
  </region-attributes>
  ```

- **Java API Configuration**

  This example shows adding two gateway senders (configured in the earlier example) to a partitioned region:

  ```
  RegionFactory rf =
    cache.createRegionFactory(RegionShortcut.PARTITION);
  rf.addCacheListener(new LoggingCacheListener());
  rf.addGatewaySenderId("sender2");
  rf.addGatewaySenderId("sender3");
  custRegion = rf.create("customer");
  ```

Note: When using the Java API, you must configure a parallel gateway sender *before* you add its id to a region. This ensures that the sender distributes region events that were persisted before new cache operations take place. If the gateway sender id does not exist when you add it to a region, you receive an `IllegalStateException`.

Configure Gateway Receivers

Always configure a gateway receiver in each VMware GemFire cluster that will receive and apply region events from another cluster.

A gateway receiver configuration can be applied to multiple VMware GemFire servers for load balancing and high availability. However, each VMware GemFire member that hosts a gateway receiver must also define all of the regions for which the receiver may receive an event. If a gateway receiver receives an event for a region that the local member does not define, VMware GemFire throws an exception. See Create Data Regions for Multi-site Communication.

Note: You can only host one gateway receiver per member.

A gateway receiver configuration specifies a range of possible port numbers on which to listen. The VMware GemFire server picks an unused port number from the specified range to use for the receiver process. You can use this functionality to easily deploy the same gateway receiver configuration to multiple members.

You can optionally configure gateway receivers to provide a specific IP address or host name for gateway sender connections. If you configure hostname-for-senders, locators will use the provided host name or IP address when instructing gateway senders on how to connect to gateway receivers. If you provide "" or null as the value, by default the gateway receiver's bind-address will be sent to clients.

In addition, you can configure gateway receivers to start automatically or, by setting `manual-start` to true, to require a manual start. By default, gateway receivers start automatically.

Note: To configure a gateway receiver, you can use gfsh, cache.xml or Java API configurations as described below. For more information on configuring gateway receivers in gfsh, see create gateway-receiver.

- **gfsh configuration command**

  ```
  gfsh>create gateway-receiver --start-port=1530 --end-port=1551 \
      --hostname-for-senders=gateway1.mycompany.com
  ```

- **cache.xml Configuration**

  The following configuration defines a gateway receiver that listens on an unused port in the range from 1530 to 1550:

  ```
  <cache>
    <gateway-receiver start-port="1530" end-port="1551"
                      hostname-for-senders="gateway1.mycompany.com" />
     ...
  </cache>
  ```

- **Java API Configuration**

```
// Create or obtain the cache
Cache cache = new CacheFactory().create();

// Configure and create the gateway receiver
GatewayReceiverFactory gateway = cache.createGatewayReceiverFactory();
gateway.setStartPort(1530);
gateway.setEndPort(1551);
gateway.setHostnameForSenders("gateway1.mycompany.com");
GatewayReceiver receiver = gateway.create();
```

Note: When using the Java API, you must create any region that might receive events from a remote site before you create the gateway receiver. Otherwise, batches of events could arrive from remote sites before the regions for those events have been created. If this occurs, the local site will throw exceptions because the receiving region does not yet exist. If you define regions in `cache.xml`, the correct startup order is handled automatically.

After starting new gateway receivers, you can execute the load-balance gateway-sender command in `gfsh` so that a specific gateway sender will be able to rebalance its connections and connect new remote gateway receivers. Invoking this command redistributes gateway sender connections more evenly among all the gateway receivers.

Another option is to use the `GatewaySender.rebalance` Java API.

As an example, assume the following scenario:

1. Create 1 receiver in site NY.
2. Create 4 senders in site LN.
3. Create 3 additional receivers in NY.

You can then execute the following in gfsh to see the effects of rebalancing:

```
gfsh -e "connect --locator=localhost[10331]" -e "list gateways"
...
(2) Executing - list gateways

GatewaySender Section

GatewaySender Id |                Member                | Remote Cluster Id |   Type   |
Status   | Queued Events | Receiver Location
---------------- | -------------------------------- | ----------------- | -------- |
------- | ------------- | -----------------
ln               | mymac(ny-1:88641)<v2>:33491       | 2                 | Parallel |
Running | 0             | mymac:5037
ln               | mymac(ny-2:88705)<v3>:29329       | 2                 | Parallel |
Running | 0             | mymac:5064
ln               | mymac(ny-3:88715)<v4>:36808       | 2                 | Parallel |
Running | 0             | mymac:5132
ln               | mymac(ny-4:88724)<v5>:52993       | 2                 | Parallel |
Running | 0             | mymac:5324

GatewayReceiver Section

             Member              | Port | Sender Count | Senders Connected
-------------------------------- | ---- | ------------ | ----------------------------
-----------------------------------------------
mymac(ny-1:88641)<v2>:33491       | 5057 | 24           | ["mymac(ln-1:88651)<v2>:4827
7","mymac(ln-4:88681)<v5>:42784","mymac(ln-2:88662)<v3>:12796","mymac(ln-3:88672)<v4>:
43675"]
mymac(ny-2:88705)<v3>:29329       | 5082 | 0            | []
mymac(ny-3:88715)<v4>:36808       | 5371 | 0            | []
mymac(ny-4:88724)<v5>:52993       | 5247 | 0            | []
```

Execute the load-balance command:

```
gfsh -e "connect --locator=localhost[10441]" -e "load-balance gateway-sender --id=n
y"...

(2) Executing - load-balance gateway-sender --id=ny

             Member               | Result | Message
--------------------------------- | ------ |-------------------------------------------
--------------------------------
mymac(ln-1:88651)<v2>:48277       | OK     | GatewaySender ny is rebalanced on member
mymac(ln-1:88651)<v2>:48277
mymac(ln-4:88681)<v5>:42784       | OK     | GatewaySender ny is rebalanced on member
mymac(ln-4:88681)<v5>:42784
mymac(ln-3:88672)<v4>:43675       | OK     | GatewaySender ny is rebalanced on member
mymac(ln-3:88672)<v4>:43675
mymac(ln-2:88662)<v3>:12796       | OK     | GatewaySender ny is rebalanced on member
mymac(ln-2:88662)<v3>:12796
```

Listing gateways in ny again shows the connections are spread much better among the receivers.

```
gfsh -e "connect --locator=localhost[10331]" -e "list gateways"...

(2) Executing - list gateways

GatewaySender Section

GatewaySender Id |              Member              | Remote Cluster Id |  Type    |
Status  | Queued Events | Receiver Location
---------------- | -------------------------------- |  ---------------- | -------- |
------- | ------------- | -----------------
ln               | mymac(ny-1:88641)<v2>:33491       | 2                 | Parallel |
Running | 0             | mymac:5037
ln               | mymac(ny-2:88705)<v3>:29329       | 2                 | Parallel |
Running | 0             | mymac:5064
ln               | mymac(ny-3:88715)<v4>:36808       | 2                 | Parallel |
Running | 0             | mymac:5132
ln               | mymac(ny-4:88724)<v5>:52993       | 2                 | Parallel |
Running | 0             | mymac:5324

GatewayReceiver Section

         Member                   | Port | Sender Count | Senders Connected
--------------------------------- | ---- | ------------ | ----------------------------
-------------------------------------------------------------------------------------
-------------------------------
mymac(ny-1:88641)<v2>:33491       | 5057 | 9            |["mymac(ln-1:88651)<v2>:4827
7","mymac(ln-4:88681)<v5>:42784","mymac(ln-3:88672)<v4>:43675","mymac(ln-2:88662)<v3>:
12796"]
mymac(ny-2:88705)<v3>:29329       | 5082 | 4            |["mymac(ln-1:88651)<v2>:4827
7","mymac(ln-4:88681)<v5>:42784","mymac(ln-3:88672)<v4>:43675"]
mymac(ny-3:88715)<v4>:36808       | 5371 | 4            |["mymac(ln-1:88651)<v2>:4827
7","mymac(ln-4:88681)<v5>:42784","mymac(ln-3:88672)<v4>:43675"]
mymac(ny-4:88724)<v5>:52993       | 5247 | 3            |["mymac(ln-1:88651)<v2>:4827
7","mymac(ln-4:88681)<v5>:42784","mymac(ln-3:88672)<v4>:43675"]
```

Running the load balance command in site ln again produces even better balance.

```
        Member                      | Port | Sender Count | Senders Connected
--------------------------------- | ---- | ------------ |-----------------------------
---------------------------------------------------------------------------------------
------------------------------
mymac(ny-1:88641)<v2>:33491       | 5057 | 7            |["mymac(ln-1:88651)<v2>:4827
7","mymac(ln-4:88681)<v5>:42784","mymac(ln-2:88662)<v3>:12796","mymac(ln-3:88672)<v4>:
43675"]
mymac(ny-2:88705)<v3>:29329       | 5082 | 3            |["mymac(ln-1:88651)<v2>:4827
7","mymac(ln-3:88672)<v4>:43675","mymac(ln-2:88662)<v3>:12796"]
mymac(ny-3:88715)<v4>:36808       | 5371 | 5            |["mymac(ln-1:88651)<v2>:4827
7","mymac(ln-4:88681)<v5>:42784","mymac(ln-2:88662)<v3>:12796","mymac(ln-3:88672)<v4>:
43675"]
mymac(ny-4:88724)<v5>:52993       | 5247 | 6            |["mymac(ln-1:88651)<v2>:4827
7","mymac(ln-4:88681)<v5>:42784","mymac(ln-2:88662)<v3>:12796","mymac(ln-3:88672)<v4>:
43675"]
```

Configuring One IP Address and Port to Access All Gateway Receivers in a Site

You may have a WAN deployment in which you do not want to expose the IP address and port of every gateway receiver to other sites, but instead expose just one IP address and port for all gateway receivers. This way, the internal topology of the site is hidden to other sites. This case is quite common in cloud deployments, in which a reverse proxy/load balancer distributes incoming requests to the site (in this case, replication requests) among the available servers (in this case, gateway receivers).

VMware GemFire supports this configuration by means of a particular use of the `hostname-for-senders`, `start-port` and `end-port` parameters of the gateway receiver.

In order to configure a WAN deployment that hides the gateway receivers behind the same IP address and port,

- All the gateway receivers must have the same value for the `hostname-for-senders` parameter (the hostname or IP address to be used by clients to access them)
- All gateway receivers must have the same value in the `start-port` and `end-port` parameters (the ports to be used by clients to access them).

The following example shows a deployment in which all gateway receivers of a site are accessed via the “gateway1.mycompany.com” hostname and port “1971”; every gateway receiver in the site must be configured as follows:

```
gfsh> create gateway-receiver --hostname-for-senders="gateway1.mycompany.com" --start-
port=1971 --end-port=1971
```

The following output shows how the receiver side would look like after this configuration if four gateway receivers were configured:

```
Cluster-ny gfsh>list gateways
GatewayReceiver Section

                    Member        | Port | Sender Count | Senders Connected
----------------------------------| ---- | ------------ | ---------------------------------------------------------------------------------------------------------------------------------
192.168.1.20(ny1:21901)<v1>:41000 | 1971 | 1            | 192.168.0.13(ln4:22520)<v4>:41005
192.168.2.20(ny2:22150)<v2>:41000 | 1971 | 2            | 192.168.0.13(ln2:22004)<v2>:41003, 192.168.0.13(ln3:22252)<v3>:41004
192.168.3.20(ny3:22371)<v3>:41000 | 1971 | 2            | 192.168.0.13(ln3:22252)<v3>:41004, 192.168.0.13(ln2:22004)<v2>:41003
192.168.4.20(ny4:22615)<v4>:41000 | 1971 | 3            | 192.168.0.13(ln4:22520)<v4>:41005, 192.168.0.13(ln1:21755)<v1>:41002, 192.168.0.13(ln1:21755)<v1>:41002
```

When the gateway senders on one site are started, they get the information about the gateway receivers of the remote site from the locator(s) running on the remote site. The remote locator provides a list of gateway receivers to send replication events to (one element per gateway receiver running in the site), with all of them listening on the same hostname and port. From the gateway sender's standpoint, it is as if only one gateway receiver is on the other side.

The following output shows the gateways information at the sender side, in which it can be seen that there is only one IP address/hostname and port for the receiver location (gateway1.mycompany.com:1971), while the reality is that there are four gateway receivers on the other side.

```
Cluster-ln gfsh>list gateways
GatewaySender Section

GatewaySender Id |                    Member         | Remote Cluster Id |   Type   | Status          | Queued Events | Receiver Location
---------------- | ----------------------------------| ----------------- | -------- | --------------------- | ------------- | ---------------------------
ny               | 192.168.0.13(ln2:22004)<v2>:41003 | 2                 | Parallel | Running and Connected | 0             | gateway1.mycompany.com:1971
ny               | 192.168.0.13(ln3:22252)<v3>:41004 | 2                 | Parallel | Running and Connected | 0             | gateway1.mycompany.com:1971
ny               | 192.168.0.13(ln4:22520)<v4>:41005 | 2                 | Parallel | Running and Connected | 0             | gateway1.mycompany.com:1971
ny               | 192.168.0.13(ln1:21755)<v1>:41002 | 2                 | Parallel | Running and Connected | 0             | gateway1.mycompany.com:1971
```

In order for the gateway senders to communicate with the remote gateway receivers, a reverse proxy/load balancer service must be in place in the deployment in order to receive the requests directed to the gateway receivers on the IP address and port configured, and to forward the requests to one of the gateway receivers on the remote site.

Filtering Events for Multi-Site (WAN) Distribution

You can optionally create gateway sender and/or gateway receiver filters to control which events are queued and distributed to a remote site, or to modify the data stream that is transmitted between VMware GemFire sites.

You can implement and deploy two different types of filter for multi-site events:

- `GatewayEventFilter`. A `GatewayEventFilter` implementation determines whether a region event is placed in a gateway sender queue and/or whether an event in a gateway queue is distributed to a remote site. You can optionally add one or more `GatewayEventFilter` implementations to a gateway sender, etiher in the `cache.xml` configuration file or using the Java API.

  VMware GemFire makes a synchronous call to the filter's `beforeEnqueue` method before it places a region event in the gateway sender queue. The filter returns a boolean value that specifies whether the event should be added to the queue.

  VMware GemFire asynchronously calls the filter's `beforeTransmit` method to determine whether the gateway sender dispatcher thread should distribute the event to a remote gateway receiver.

  For events that are distributed to another site, VMware GemFire calls the listener's `afterAcknowledgement` method to indicate that is has received an ack from the remote site after the event was received.

- GatewayTransportFilter. Use a `GatewayTransportFilter` implementation to process the TCP stream that sends a batch of events that is distributed from one VMware GemFire cluster to another over a WAN. A `GatewayTransportFilter` is typically used to perform encryption or compression on the data that distributed. You install the same `GatewayTransportFilter` implementation on both a gateway sender and gateway receiver.

  When a gateway sender processes a batch of events for distribution, VMware GemFire delivers the stream to the `getInputStream` method of a configured `GatewayTransportFilter` implementation. The filter processes and returns the stream, which is then transmitted to the gateway receiver. When the gateway receiver receives the batch, VMware GemFire calls the `getOutputStream` method of a configured filter, which again processes and returns the stream so that the events can be applied in the local cluster.

Configuring Multi-Site Event Filters

You install a `GatewayEventFilter` implementation to a configured gateway sender in order to decide which events are queued and distributed. You install a `GatewayTransportFilter` implementation to both a gateway sender and a gateway receiver to process the stream of batched events that are distributed between two sites:

- **XML example**

```
<cache>
  <gateway-sender id="remoteA" parallel="true" remote-distributed-system-id
="1">
    <gateway-event-filter>
      <class-name>org.apache.geode.util.SampleEventFilter</class-name>
      <parameter name="param1">
        <string>"value1"</string>
      </parameter>
    </gateway-event-filter>
    <gateway-transport-filter>
      <class-name>org.apache.geode.util.SampleTransportFilter</class-name>
      <parameter name="param1">
```

```
        <string>"value1"</string>
      </parameter>
    </gateway-transport-filter>
  </gateway-sender>
</cache>
```

```
<cache>
  ...
  <gateway-receiver start-port="1530" end-port="1551">
    <gateway-transport-filter>
      <class-name>org.apache.geode.util.SampleTransportFilter</class-name>
      <parameter name="param1">
        <string>"value1"</string>
      </parameter>
    </gateway-transport-filter>
  </gateway-receiver>
</cache>
```

- **gfsh example**

  ```
  gfsh>create gateway-sender --id=remoteA --parallel=true --remote-distributed-id
  ="1"
  --gateway-event-filter=org.apache.geode.util.SampleEventFilter
  --gateway-transport-filter=org.apache.geode.util.SampleTransportFilter
  ```

  See create gateway-sender.

  ```
  gfsh>create gateway-receiver --start-port=1530 --end-port=1551 \
  --gateway-transport-filter=org.apache.geode.util.SampleTransportFilter
  ```

  Note: You cannot specify parameters and values for the Java class you specify with the `--gateway-transport-filter` option.

  See create gateway-receiver.

- **API example**

  ```
  Cache cache = new CacheFactory().create();

  GatewayEventFilter efilter = new SampleEventFilter();
  GatewayTransportFilter tfilter = new SampleTransportFilter();

  GatewaySenderFactory gateway = cache.createGatewaySenderFactory();
  gateway.setParallel(true);
  gateway.addGatewayEventFilter(efilter);
  gateway.addTransportFilter(tfilter);
  GatewaySender sender = gateway.create("remoteA", "1");
  sender.start();
  ```

  ```
  Cache cache = new CacheFactory().create();

  GatewayTransportFilter tfilter = new SampleTransportFilter();

  GatewayReceiverFactory gateway = cache.createGatewayReceiverFactory();
  gateway.setStartPort(1530);
  gateway.setEndPort(1551);
  ```

```
gateway.addTransportFilter(tfilter);
GatewayReceiver receiver = gateway.create();
receiver.start();
```

Resolving Conflicting Events

You can optionally create a `GatewayConflictResolver` cache plug-in to decide whether a potentially conflicting event that was delivered from another site should be applied to the local cache.

By default, all regions perform consistency checks when a member applies an update received either from another cluster member or from a remote cluster over the WAN. The default consistency checking for WAN events is described in How Consistency Is Achieved in WAN Deployments.

You can override the default consistency checking behavior by writing and configuring a custom `GatewayConflictResolver`. The `GatewayConflictResolver` implementation can use the timestamp and distributed system ID included in a WAN update event to determine whether or not to apply the update. For example, you may decide that updates from a particular cluster should always "win" a conflict when the timestamp difference between updates is less than some fixed period of time.

Implementing a GatewayConflictResolver

Note: A `GatewayConflictResolver` implementation is called only for update events that could cause a conflict in the region. This corresponds to update events that have a different distributed system ID than the distributed system that last updated the region entry. If the same distributed system ID makes consecutive updates to a region entry, no conflict is possible, and the `GatewayConflictResolver` is not called.

Procedure

1. Program the event handler:
    1. Create a class that implements the `GatewayConflictResolver` interface.
    2. If you want to declare the handler in `cache.xml`, implement the `org.apache.geode.cache.Declarable` interface as well.
    3. Implement the handler's `onEvent()` method to determine whether the WAN event should be allowed. `onEvent()` receives both a `TimestampedEntryEvent` and a `GatewayConflictHelper` instance. `TimestampedEntryEvent` has methods for obtaining the timestamp and distributed system ID of both the update event and the current region entry. Use methods in the `GatewayConflictHelper` to either disallow the update event (retaining the existing region entry value) or provide an alternate value.

       Example:

       ```
        public void onEvent(TimestampedEntryEvent event, GatewayConflictHelper h
       elper) {
           if (event.getOperation().isUpdate()) {
             ShoppingCart oldCart = (ShoppingCart)event.getOldValue();
             ShoppingCart newCart = (ShoppingCart)event.getNewValue();
             oldCart.updateFromConflictingState(newCart);
       ```

```
      helper.changeEventValue(oldCart);
    }
  }
```

Note: In order to maintain consistency in the region, your conflict resolver must always resolve two events in the same way regardless of which event it receives first.

2. Install the conflict resolver for the cache, using either the `cache.xml` file or the Java API.

   cache.xml

   ```
   <cache>
        ...
       <gateway-conflict-resolver>
         <class-name>myPackage.MyConflictResolver</class-name>
       </gateway-conflict-resolver>
       ...
   </cache>
   ```

   Java API

   ```
   // Create or obtain the cache
   Cache cache = new CacheFactory().create();

   // Create and add a conflict resolver
   cache.setGatewayConflictResolver(new MyConflictResolver);
   ```

Managing VMware GemFire

Managing VMware GemFire describes how to plan and implement tasks associated with managing, monitoring, and troubleshooting VMware GemFire.

- **VMware GemFire Management and Monitoring**

  VMware GemFire provides APIs and tools for managing your distributed system and monitoring the health of your distributed system members.

- **Managing Heap and Off-heap Memory**

  By default, VMware GemFire uses the JVM heap. VMware GemFire also offers an option to store data off heap. This section describes how to manage heap and off-heap memory to best support your application.

- **Disk Storage**

  With VMware GemFire disk stores, you can persist data to disk as a backup to your in-memory copy and overflow data to disk when memory use gets too high.

- **Cache and Region Snapshots**

  Snapshots allow you to save region data and reload it later. A typical use case is loading data from one environment into another, such as capturing data from a production system and moving it into a smaller QA or development system.

- **Region Compression**

  This section describes region compression, its benefits and usage.

- **Network Partitioning**

  VMware GemFire architecture and management features help detect and resolve network partition problems.

- **Security**

  The security framework establishes trust by authenticating components and members upon connection. It facilitates the authorization of operations.

- **Performance Tuning and Configuration**

  A collection of tools and controls allow you to monitor and adjust VMware GemFire performance.

- **Logging**

  Comprehensive logging messages help you confirm system configuration and debug problems in configuration and code.

- **Statistics**

Every application and server in a distributed system can access statistical data about VMware GemFire operations. You can configure the gathering of statistics by using the `alter runtime` command of `gfsh` or in the `gemfire.properties` file to facilitate system analysis and troubleshooting.

- **Troubleshooting and System Recovery**

  This section provides strategies for handling common errors and failure situations.

VMware GemFire Management and Monitoring

VMware GemFire provides APIs and tools for managing your cluster and monitoring the health of your members.

- **Management and Monitoring Features**

  VMware GemFire uses a federated Open MBean strategy to manage and monitor all members of the cluster. This strategy gives you a consolidated, single-agent view of the cluster.

- **Overview of VMware GemFire Management and Monitoring Tools**

  VMware GemFire provides a variety of management tools you can use to manage a VMware GemFire cluster.

- **Architecture and Components**

  VMware GemFire's management and monitoring system consists of one JMX Manager node (there should only be one) and one or more managed nodes within a cluster. All members in the cluster are manageable through MBeans and VMware GemFire Management Service APIs.

- **JMX Manager Operations**

  Any member can host an embedded JMX Manager, which provides a federated view of all MBeans for the cluster. The member can be configured to be a manager at startup or anytime during its life by invoking the appropriate API calls on the ManagementService.

- **Federated MBean Architecture**

  VMware GemFire uses MBeans to manage and monitor different parts of VMware GemFire. VMware GemFire's federated MBean architecture is scalable and allows you to have a single-agent view of a VMware GemFire cluster.

- **Configuring RMI Registry Ports and RMI Connectors**

  VMware GemFire programmatically emulates out-of-the-box JMX provided by Java and creates a JMXServiceURL with RMI Registry and RMI Connector ports on all manageable members.

- **Executing gfsh Commands through the Management API**

  You can also use management APIs to execute gfsh commands programmatically.

Management and Monitoring Features

VMware GemFire uses a federated Open MBean strategy to manage and monitor all members of the cluster. This strategy gives you a consolidated, single-agent view of the cluster.

Application and manager development is much easier because you do not have to find the right MBeanServer to make a request on an MBean. Instead, you interact with a single MBeanServer that aggregates MBeans from all other local and remote MBeanServers.

Some other key advantages and features of VMware GemFire administration architecture:

- VMware GemFire monitoring is tightly integrated into VMware GemFire's processes instead of running in a separately installed and configured monitoring agent. You can use the same framework to actually manage VMware GemFire and perform administrative operations, not just monitor it.
- All VMware GemFire MBeans are *MXBeans*. They represent useful and relevant information on the state of the cluster and all its members. Because MXBeans use the Open MBean model with a predefined set of types, clients and remote management programs no longer require access to model-specific classes representing your MBean types. Using MXBeans adds flexibility to your selection of clients and makes the VMware GemFire management and monitoring much easier to use.
- Each member in the cluster is manageable through MXBeans, and each member hosts its own MXBeans in a Platform MBeanServer.
- Any VMware GemFire member can be configured to provide a federated view of all the MXBeans for all members in a VMware GemFire cluster.
- VMware GemFire has also modified its use of JMX to be industry-standard and friendly to generic JMX clients. You can now easily monitor or manage the cluster by using any third-party tool that is compliant with JMX. For example, JConsole.

References

For more information on MXBeans and Open MBeans, see:

- http://docs.oracle.com/javase/8/docs/api/javax/management/MXBean.html
- http://docs.oracle.com/javase/8/docs/api/javax/management/openmbean/package-summary.html

Overview of VMware GemFire Management and Monitoring Tools

VMware GemFire provides a variety of management tools you can use to manage a VMware GemFire cluster.

The VMware GemFire management and monitoring tools allow you to configure all members and processes of a cluster, monitor operations in the system, and start and stop the members. Internally, VMware GemFire uses Java MBeans, specifically MXBeans, to expose management controls and monitoring features. You can monitor and control VMware GemFire by writing Java programs that use these MXBeans, or you can use one of several tools provided with VMware GemFire to monitor and manage your cluster. The primary tool for these tasks is the gfsh command-line tool, as described in this section.

VMware GemFire provides the following tools to manage a VMware GemFire installation:

gfsh Command-line tool

The gfsh command line tool provides a set of commands you use to configure, manage, and monitor a cluster. gfsh is the recommended tool for managing your cluster.

Use gfsh to:

- Start and stop VMware GemFire processes, such as locators and cache servers
- Deploy applications
- Create and destroy regions
- Execute functions
- Manage disk stores
- Import and export data
- Monitor VMware GemFire processes
- Launch VMware GemFire monitoring tools
- Shut down a cluster
- Script various operations involving VMware GemFire members
- Save the configuration for all members of a cluster

gfsh runs in its own shell, or you can execute gfsh commands directly from the OS command line. gfsh can interact with remote systems using the http protocol. You can also write scripts that run in a gfsh shell to automate system startup.

You can use gfsh to create shared cluster configurations for your cluster. You can define configurations that apply to the entire cluster, or that apply only to groups of similar members that all share a common configuration. VMware GemFire locators maintain these configurations as a hidden region and distribute the configuration to all locators in the cluster. The locator also persists the shared configurations on disk as `cluster.xml` and `cluster.properties` files. You can use those shared cluster configuration files to re-start your system, migrate the system to a new environment, add new members to a cluster, or to restore existing members after a failure.

A basic cluster configuration consists of:

- `cluster.xml` file shared by the cluster
- `cluster.properties` file shared by the cluster
- Deployed jar files containing application Java classes.

See Overview of the Cluster Configuration Service and Cluster Configuration Files and Troubleshooting for additional details on gfsh cluster configuration files.

Using the gfsh tool, you can easily migrate a VMware GemFire-based application from a development environment into a testing or production environment.

Executing gfsh commands with the management API

You can also use VMware GemFire's management APIs to execute gfsh commands in a Java class. See Executing gfsh Commands through the Management API.

Member Configuration Management

When you issue gfsh commands and have the cluster configuration service enabled (on a locator), VMware GemFire saves the configurations created within gfsh by building a `cluster.xml` and `cluster.properties` files for the entire cluster, or group of members.

You can also directly create configurations using `cache.xml` and `gemfire.properties` files and manage the members individually.

Java Management Extension (JMX) MBeans

VMware GemFire uses a federated Open MBean strategy to manage and monitor all members of the cluster. Your Java classes interact with a single MBeanServer that aggregates MBeans from other local and remote members. Using this strategy gives you a consolidated, single-agent view of the cluster.

VMware GemFire's implementation of JMX is industry-standard and friendly to generic JMX clients. You can monitor or manage the cluster by using any third-party tool that is compliant with JMX. For example, JConsole.

See VMware GemFire Management and Monitoring

VMware GemFire Java API

The VMware GemFire API provides a set of Java classes you can use to manage and monitor a cluster. See the `org.apache.geode.management` package in the javadocs.

VMware GemFire Pulse

VMware GemFire Pulse is a Web Application that provides a graphical dashboard for monitoring vital, real-time health and performance of VMware GemFire clusters, members, and regions.

Use Pulse to examine total memory, CPU, and disk space used by members, uptime statistics, client connections, and critical notifications. Pulse communicates with a VMware GemFire JMX manager to provide a complete view of your VMware GemFire deployment.

See VMware GemFire Pulse.

JConsole

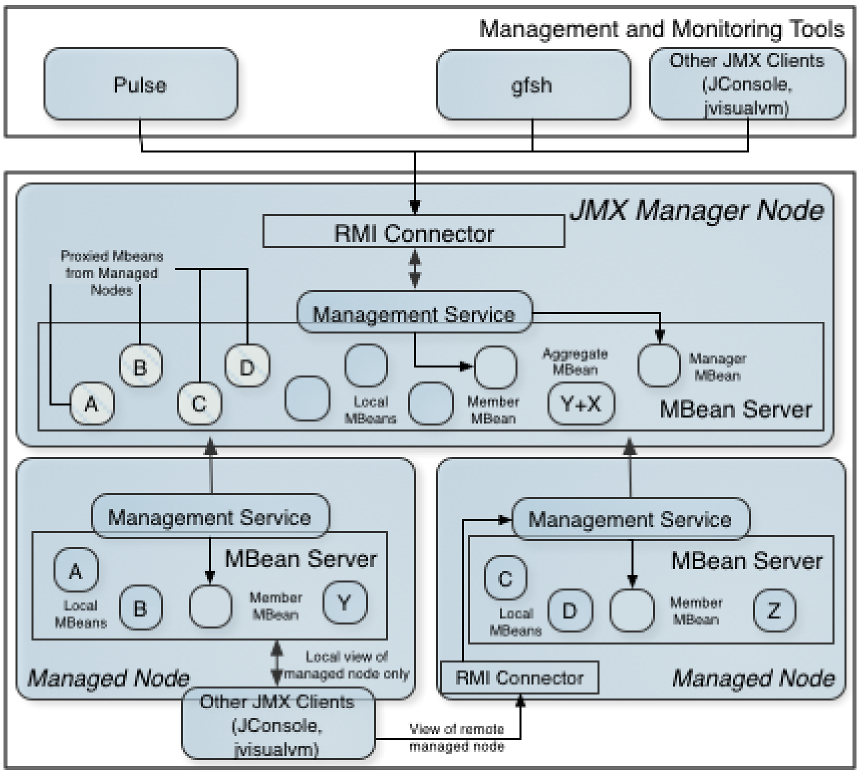
JConsole is a JMX monitoring utility provided with a Java Development Kit (JDK). You use gfsh to connect to VMware GemFire, and then launch JConsole with a gfsh command. The JConsole application allows you to browse MBeans, attributes, operations, and notifications. See Browsing VMware GemFire MBeans through JConsole.

Architecture and Components

VMware GemFire's management and monitoring system consists of one JMX Manager node (there should only be one) and one or more managed nodes within a cluster. All members in the cluster are manageable through MBeans and VMware GemFire Management Service APIs.

Architecture

The following diagram depicts the architecture of the management and monitoring system components.



In this architecture every VMware GemFire member is manageable. All VMware GemFire MBeans for the local VMware GemFire processes are automatically registered in the Platform MBeanServer (the default MBeanServer of each JVM that hosts platform MXBeans.)

Managed Node

Each member of a cluster is a managed node. Any node that is not currently also acting as a JMX Manager node is referred to simply as a managed node. A managed node has the following resources so that it can answer JMX queries both locally and remotely:

- Local MXBeans that represent the locally monitored components on the node. See List of VMware GemFire JMX MBeans for a list of possible MXBeans existing for the managed node.
- Built-in platform MBeans.

JMX Manager Node

A JMX Manager node is a member that can manage other VMware GemFire members—that is, other managed nodes—as well as itself. A JMX Manager node can manage all other members in the cluster.

To convert a managed node to a JMX Manager node, you configure the VMware GemFire property `jmx-manager=true`, in the `gemfire.properties` file, and start the member as a JMX Manager node.

You start the member as a JMX Manager node when you provide `--J=-Dgemfire.jmx-manager=true` as an argument to either the `start server` or `start locator` command. See Starting a JMX Manager for more information.

The JMX Manager node has the following extra resources allocated so that it can answer JMX queries:

- RMI connector that allows JMX clients to connect to and access all MXBeans in the cluster.
- Local MXBeans that represent the locally monitored components on this node, same as any other managed node.
- Aggregate MXBeans:
  - DistributedSystemMXBean
  - DistributedRegionMXBean
  - DistributedLockServiceMXBean
- ManagerMXBean with Scope=ALL, which allows various cluster-wide operations.
- Proxy to MXBeans on managed nodes.
- Built-in platform MXBeans.

JMX Integration

Management and monitoring tools such as gfsh command-line interface and Pulse use JMX/RMI as the communication layer to connect to VMware GemFire nodes. All VMware GemFire processes by default allow JMX connections to the Platform MBeanServer from localhost. By default, both managed nodes and JMX manager nodes have RMI connectors enabled to allow JMX client connections.

JConsole (and other similar JMX clients that support Sun's Attach API) can connect to any local JVM without requiring an RMI connector by using the Attach API. This allows connections from the same machine.

JConsole (and other JMX clients) can connect to any JVM if that JVM is configured to start an RMI connector. This allows remote connections from other machines.

JConsole can connect to any VMware GemFire member, but if it connects to a non-JMX-Manager member, JConsole only detects the local MBeans for the node, and not MBeans for the cluster.

When a VMware GemFire locator or server becomes a JMX Manager for the cluster, it enables the RMI connector. JConsole can then connect only to that one JVM to view the MBeans for the entire cluster. It does not need to connect to all the other JVMs. VMware GemFire manages the inter-JVM communication required to provide a federated view of all MBeans in the cluster.

`gfsh` can only connect to a JMX Manager or to a locator. If connected to a locator, the locator provides the necessary connection information for the existing JMX Manager. If the locator detects a JMX Manager is not already running in the cluster, the locator makes itself a JMX Manager. gfsh cannot connect to other non-Manager or non-locator members.

For information on how to configure the RMI registry and RMI connector, see Configuring RMI Registry Ports and RMI Connectors.

Management APIs

VMware GemFire management APIs represent the VMware GemFire cluster to a JMX user. However, they do not provide functionality that is otherwise present in JMX. They only provide a gateway into various services exclusively offered by VMware GemFire monitoring and management.

The entry point to VMware GemFire management is through the ManagementService interface. For example, to create an instance of the Management Service:

```
ManagementService service = ManagementService.getManagementService(cache);
```

The resulting ManagementService instance is specific to the provided cache and its cluster. The implementation of getManagementService is a singleton for now but may eventually support multiple cache instances.

You can use the VMware GemFire management APIs to accomplish the following tasks:

- Monitor the health status of clients.
- Obtain the status and results of individual disk backups.
- View metrics related to disk usage and performance for a particular member.
- Browse VMware GemFire properties set for a particular member.
- View JVM metrics such as memory, heap, and thread usage.
- View network metrics, such as bytes received and sent.
- View partition region attributes such as total number of buckets, redundant copy, and maximum memory information.
- View persistent member information such as disk store ID.
- Browse region attributes.

See the JavaDocs for the `org.apache.geode.management` package for more details.

You can also execute gfsh commands using the ManagementService API. See Executing gfsh Commands through the Management API and the JavaDocs for the `org.apache.geode.management.cli` package.

VMware GemFire Management and Monitoring Tools

This section lists the currently available tools for managing and monitoring VMware GemFire:

- **gfsh**. VMware GemFire command-line interface that provides a simple & powerful command shell that supports the administration, debugging and deployment of VMware GemFire applications. It features context sensitive help, scripting and the ability to invoke any commands from within the application using a simple API. See gfsh.
- **VMware GemFire Pulse**. Easy-to-use, browser-based dashboard for monitoring VMware GemFire deployments. VMware GemFire Pulse provides an integrated view of all VMware

GemFire members within a cluster. See VMware GemFire Pulse.

- **Pulse Data Browser**. This VMware GemFire Pulse utility provides a graphical interface for performing OQL ad-hoc queries in a VMware GemFire cluster. See Data Browser.
- **Other Java Monitoring Tools such as JConsole and jvisualvm.** JConsole is a JMX-based management and monitoring tool provided in the Java 2 Platform that provides information on the performance and consumption of resources by Java applications. See http://docs.oracle.com/javase/6/docs/technotes/guides/management/jconsole.html. **Java VisualVM (jvisualvm)** is a profiling tool for analyzing your Java Virtual Machine. Java VisualVM is useful to Java application developers to troubleshoot applications and to monitor and improve the applications' performance. Java VisualVM can allow developers to generate and analyse heap dumps, track down memory leaks, perform and monitor garbage collection, and perform lightweight memory and CPU profiling. For more details on using jvisualvm, see http://docs.oracle.com/javase/6/docs/technotes/tools/share/jvisualvm.html.

Starting a JMX Manager

JMX Manager nodes are members that manage other VMware GemFire members (as well as themselves). A JMX Manager node can manage all other members in the cluster. Typically a locator will function as the JMX Manager, but you can also turn any other member such as a server into a JMX Manager node as well.

To allow a server to become a JMX Manager you configure VMware GemFire property `jmx-manager=true`, in the server's`gemfire.properties` file. This property configures the node to become a JMX Manager node passively; if gfsh cannot locate a JMX Manager when connecting to the cluster, the server node will be started as a JMX Manager node.

Note: The default property setting for all locators is `gemfire.jmx-manager=true`. For other members, the default property setting is `gemfire.jmx-manager=false`.

To force a server to become a JMX Manager node whenever it is started, set the VMware GemFire properties `jmx-manager-start=true` and `jmx-manager=true` in the server's gemfire.properties file. Note that both of these properties must be set to true for the node.

To start the member as a JMX Manager node on the command line, provide`--J=-Dgemfire.jmx-manager-start=true and --J=-Dgemfire.jmx-manager=true` as arguments to either the `start server` or `start locator` command.

For example, to start a server as a JMX Manager on the gfsh command line:

```
gfsh>start server --name=<server-name> --J=-Dgemfire.jmx-manager=true \
--J=-Dgemfire.jmx-manager-start=true
```

By default, any locator can become a JMX Manager when started. When you start up a locator, if no other JMX Manager is detected in the cluster, the locator starts one automatically. If you start a second locator, it will detect the current JMX Manager and will not start up another JMX Manager unless the second locator's `gemfire.jmx-manager-start` property is set to true.

For most deployments, you only need to have one JMX Manager per cluster. However, you can run more than one JMX Manager if necessary. If you want to provide high-availability and redundancy for the Pulse monitoring tool, or if you are running additional JMX clients other than gfsh, then use the `jmx-manager-start=true` property to force individual nodes (either locators or servers) to become JMX Managers at startup. Since there is some performance overhead to being a JMX Manager, we recommend using locators as JMX Managers. If you do not want a locator to become a JMX manager, then you must use the `jmx-manager=false` property when you start the locator.

After the node becomes a JMX Manager, all other `jmx-manager-*` configuration properties listed in Configuring a JMX Manager are applied.

The following is an example of starting a new locator that also starts an embedded JMX Manager (after detecting that another JMX Manager does not exist). In addition, `gfsh` also automatically connects you to the new JMX Manager. For example:

```
gfsh>start locator --name=locator1
Starting a VMware GemFire Locator in /Users/username/apache-geode/locator1...
....
Locator in /Users/username/apache-geode/locator1 on 192.0.2.0[10334] as locator1
is currently online.
Process ID: 27144
Uptime: 5 seconds
VMware GemFire Version: 9.10
Java Version: 1..0_272
Log File: /Users/username/apache-geode/locator1/locator1.log
JVM Arguments: -Dgemfire.enable-cluster-configuration=true
-Dgemfire.load-cluster-configuration-from-dir=false
-Dgemfire.launcher.registerSignalHandlers=true
-Djava.awt.headless=true -Dsun.rmi.dgc.server.gcInterval=9223372036854775806
Class-Path: /Users/username/apache-geode/lib/geode-core-1.2.0.jar
:/Users/username/apache-geode/lib/geode-dependencies.jar

Successfully connected to: JMX Manager [host=192.0.2.0, port=1099]

Cluster configuration service is up and running.
```

Locators also keep track of all nodes that can become a JMX Manager.

Immediately after creating its cache, the JMX Manager node begins federating the MBeans from other members. After the JMX Manager node is ready, the JMX Manager node sends a notification to all other members informing them that it is a new JMX Manager. The other members then put complete MBean states for themselves into each of their hidden management regions.

At any point, you can determine whether a node is a JMX Manager by using the MemberMXBean isManager() method.

Using the Java API, any managed node that has been configured with `jmx-manager=true` can also be turned into a JMX Manager Node by invoking the ManagementService startManager() method.

Note: If you start the JMX Manager programmatically and wish to enable command processing, you must also add the absolute path of `gfsh-dependencies.jar` (located in the `lib` directory of your installation) to the CLASSPATH of your application. Do not copy this library to your CLASSPATH, because this library refers to other dependencies in `lib` by a relative path.

Configuring a JMX Manager

In the `gemfire.properties` file, you configure a JMX manager as follows.

Property	Description	Default
http-service-port	If non-zero, then VMware GemFire starts an embedded HTTP service that listens on this port. The HTTP service is used to host the VMware GemFire Pulse Web application. If you are hosting the Pulse web app on your own Web server, then disable this embedded HTTP service by setting this property to zero. Ignored if `jmx-manager` is false.	7070
http-service-bind-address	If set, then the VMware GemFire member binds the embedded HTTP service to the specified address. If this property is not set but the HTTP service is enabled using `http-service-port`, then VMware GemFire binds the HTTP service to the member's local address.	*not set*
jmx-manager	If `true` then this member can become a JMX Manager. All other `jmx-manager-*` properties are used when it does become a JMX Manager. If this property is false then all other `jmx-manager-*` properties are ignored. The default value is `true` on locators.	false (with Locator exception)
jmx-manager-access-file	By default the JMX Manager allows full access to all MBeans by any client. If this property is set to the name of a file, then it can restrict clients to only reading MBeans; they cannot modify MBeans. The access level can be configured differently in this file for each user name defined in the password file. For more information about the format of this file see Oracle's documentation of the `com.sun.management.jmxremote.access.file` system property. Ignored if `jmx-manager` is false or if `jmx-manager-port` is zero.	*not set*
jmx-manager-bind-address	By default, the JMX Manager when configured with a port listens on all the local host's addresses. You can use this property to configure which particular IP address or host name the JMX Manager will listen on. This property is ignored if `jmx-manager` is false or `jmx-manager-port` is zero. This address also applies to the VMware GemFire Pulse server if you are hosting a Pulse web application.	*not set*

Property	Description	Default
jmx-manager-hostname-for-clients	Hostname given to clients that ask the locator for the location of a JMX Manager. By default the IP address of the JMX Manager is used. However, for clients on a different network, you can configure a different hostname to be given to clients. Ignored if `jmx-manager` is false or if `jmx-manager-port` is zero.	*not set*
jmx-manager-password-file	By default the JMX Manager allows clients without credentials to connect. If this property is set to the name of a file, only clients that connect with credentials that match an entry in this file will be allowed. Most JVMs require that the file is only readable by the owner. For more information about the format of this file see Oracle's documentation of the com.sun.management.jmxremote.password.file system property. Ignored if jmx-manager is false or if jmx-manager-port is zero.	*not set*
jmx-manager-port	Port on which this JMX Manager listens for client connections. If this property is set to zero, VMware GemFire does not allow remote client connections. Alternatively, use the standard system properties supported by the JVM for configuring access from remote JMX clients. Ignored if jmx-manager is false. The Default RMI port is 1099.	1099
jmx-manager-ssl-enabled	If true and `jmx-manager-port` is not zero, the JMX Manager accepts only SSL connections. The ssl-enabled property does not apply to the JMX Manager, but the other SSL properties do. This allows SSL to be configured for just the JMX Manager without needing to configure it for the other VMware GemFire connections. Ignored if `jmx-manager` is false.	false
jmx-manager-start	If true, this member starts a JMX Manager when it creates a cache. In most cases you should not set this property to true because a JMX Manager is automatically started when needed on a member that sets `jmx-manager` to true. Ignored if jmx-manager is false.	false

Property	Description	Default
jmx-manager-update-rate	The rate, in milliseconds, at which this member pushes updates to any JMX Managers. Currently this value should be greater than or equal to the `statistic-sample-rate`. Setting this value too high causes `gfsh` and VMware GemFire Pulse to see stale values.	2000

Stopping a JMX Manager

To stop a JMX Manager using gfsh, simply shut down the locator or server hosting the JMX Manager.

For a locator:

```
gfsh>stop locator --dir=locator1
Stopping Locator running in /home/user/test2/locator1 on ubuntu.local[10334] as locato
r1...
Process ID: 2081
Log File: /home/user/test2/locator1/locator1.log
....
No longer connected to ubuntu.local[1099].
```

For a server:

```
gfsh>stop server --dir=server1
Stopping Cache Server running in /home/user/test2/server1 ubuntu.local[40404] as serve
r1...
Process ID: 1156
Log File: /home/user/test2/server1/server1.log
....


No longer connected to ubuntu.local[1099].
```

Notice that `gfsh` has automatically disconnected you from the stopped JMX Manager.

To stop a JMX manager using the management API, use the ManagementService stopManager() method to stop a member from being a JMX Manager.

When a Manager stops, it removes all federated MBeans from other members from its Platform MBeanServer. It also emits a notification to inform other members that it is no longer considered a JMX Manager.

Federated MBean Architecture

VMware GemFire uses MBeans to manage and monitor different parts of VMware GemFire. VMware GemFire's federated MBean architecture is scalable and allows you to have a single-agent view of a VMware GemFire cluster.

Federation of VMware GemFire MBeans and MBeanServers

Federation of the MBeanServers means that one member, the JMX Manager Node, can provide a proxied view of all the MBeans that the MBeanServer hosts. Federation also means that operations and notifications are spread across the cluster.

VMware GemFire federation takes care of the following functionality:

- MBean proxy creation
- MBean state propagation
- Notifications propagation
- Operation invocation

MBean Proxy Naming Conventions

Each VMware GemFire MBean follows a particular naming convention for easier grouping. For example:

```
GemFire:type=Member,service=LockService,name=<dlsName>,memberName=<memberName>
```

At the JMX Manager node, this MBean will be registered with GemFire/<memberId> as domain.

The following are some sample MBean names:

MemberMBean:

```
GemFire:type=Member,member=<Node1>
```

Use of MXBeans

In its Management API, VMware GemFire provides MXBeans to ensure that any MBeans that are created are usable by any client, including remote clients, without requiring the client to access specific classes in order to access contents of the MBean.

MBean Proxy Creation

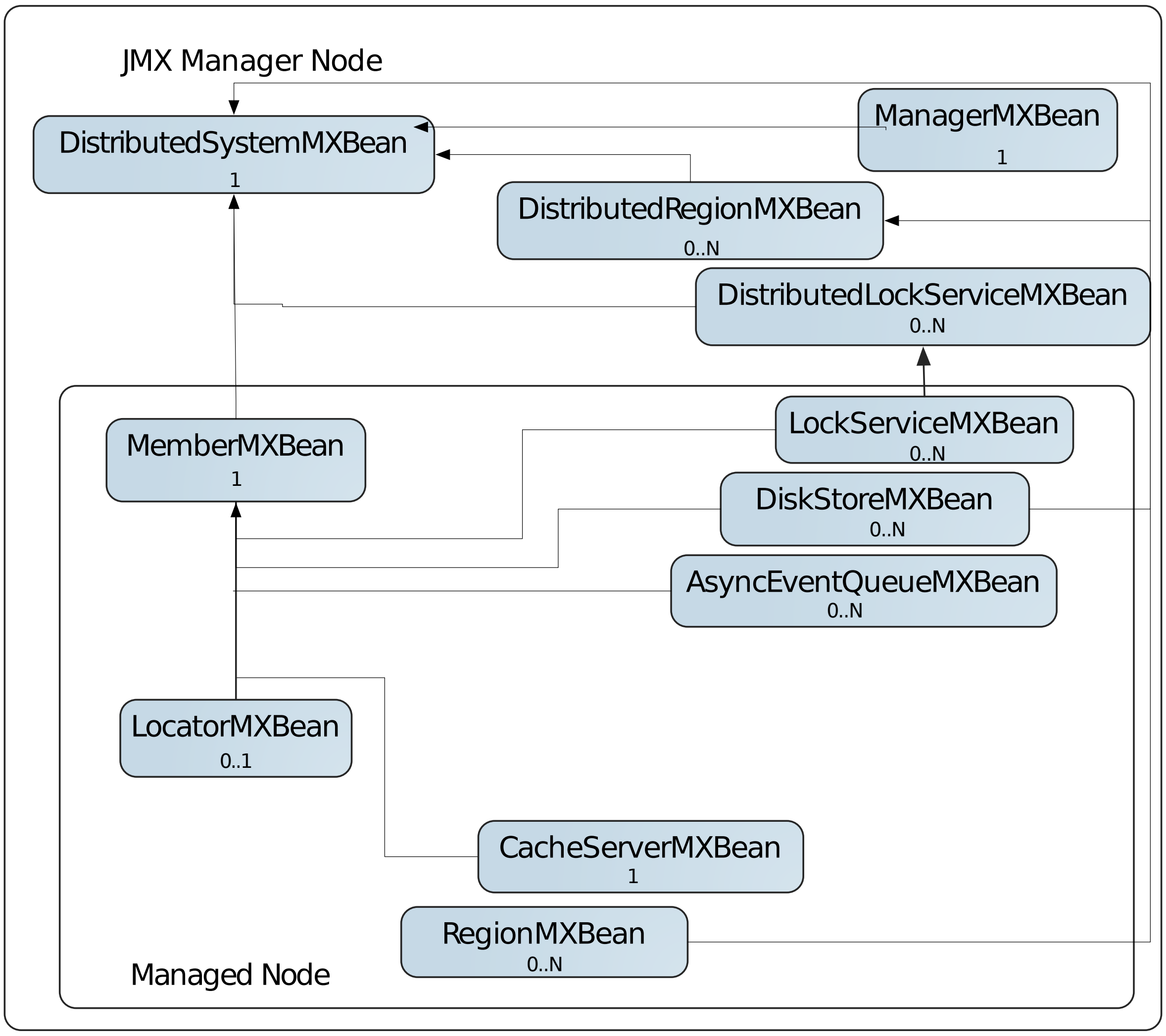
VMware GemFire proxies are inherently local MBeans. Every VMware GemFire JMX manager member hosts proxies pointing to the local MBeans of every managed node. Proxy MBeans will also emit any notification emitted by local MBeans in managed nodes when an event occurs in that managed node.

Note: Aggregate MBeans on the JMX Manager node are not proxied.

List of VMware GemFire JMX MBeans

This topic provides descriptions for the various management and monitoring MBeans that are available in VMware GemFire.

The following diagram illustrates the relationship between the different JMX MBeans that have been developed to manage and monitor VMware GemFire.



- **JMX Manager MBeans**

  This section describes the MBeans that are available on the JMX Manager node.

- **Managed Node MBeans**

  This section describes the MBeans that are available on all managed nodes.

JMX Manager MBeans

This section describes the MBeans that are available on the JMX Manager node.

The JMX Manager node includes all local beans listed under Managed Node MBeans and the following beans that are available only on the JMX Manager node:

- ManagerMXBean
- DistributedSystemMXBean
- DistributedRegionMXBean
- DistributedLockServiceMXBean

ManagerMXBean

Represents the VMware GemFire Management layer for the hosting member. Controls the scope of management. This MBean provides `start` and `stop` methods to turn a managed node into a JMX Manager node or to stop a node from being a JMX Manager. For potential managers (`jmx-manager=true` and `jmx-manager-start=false`), this MBean is created when a Locator requests it.

Note: You must configure the node to allow it to become a JMX Manager. See Configuring a JMX Manager for configuration information.

MBean Details

Scope	ALL
Proxied	No
Object Name	GemFire:type=Member, service=Manager,member=<name-or-dist-member-id>
Instances Per Node	1

See the `org.apache.geode.management.ManagerMXBean` JavaDocs for information on available MBean methods and attributes.

DistributedSystemMXBean

System-wide aggregate MBean that provides a high-level view of the entire cluster including all members (cache servers, peers, locators) and their caches. At any given point of time, it can provide a snapshot of the complete cluster and its operations.

The DistributedSystemMXBean provides APIs for performing cluster-wide operations such as backing up all members, shutting down all members or showing various cluster metrics.

You can attach a standard JMX NotificationListener to this MBean to listen for notifications throughout the cluster. See VMware GemFire JMX MBean Notifications for more information.

This MBean also provides some MBean model navigation APIS. These APIs should be used to navigate through all the MBeans exposed by a VMware GemFire System.

MBean Details

Scope	Aggregate
Proxied	No
Object Name	GemFire:type=Distributed,service=System
Instances Per Node	1

See the `org.apache.geode.management.DistributedSystemMXBean` JavaDocs for information on available MBean methods and attributes.

DistributedRegionMXBean

System-wide aggregate MBean of a named region. It provides a high-level view of a region for all members hosting and/or using that region. For example, you can obtain a list of all members that are hosting the region. Some methods are only available for partitioned regions.

MBean Details

Scope	Aggregate
Proxied	No
Object Name	GemFire:type=Distributed,service=Region,name=<regionName>
Instances Per Node	0..N

See the `org.apache.geode.management.DistributedRegionMXBean` JavaDocs for information on available MBean methods and attributes.

DistributedLockServiceMXBean

Represents a named instance of DistributedLockService . Any number of DistributedLockService can be created in a member.

A named instance of DistributedLockService defines a space for locking arbitrary names across the cluster defined by a specified distribution manager. Any number of DistributedLockService instances can be created with different service names. For all processes in the cluster that have created an instance of DistributedLockService with the same name, no more than one thread is permitted to own the lock on a given name in that instance at any point in time. Additionally, a thread can lock the entire service, preventing any other threads in the system from locking the service or any names in the service.

MBean Details

Scope	Aggregate
Proxied	No
Object Name	GemFire:type=Distributed,service=LockService,name=<dlsName>
Instances Per Node	0..N

See the `org.apache.geode.management.DistributedLockServiceMXBean` JavaDocs for information on available MBean methods and attributes.

Managed Node MBeans

This section describes the MBeans that are available on all managed nodes.

MBeans that are available on all managed nodes include:

- MemberMXBean
- CacheServerMXBean
- RegionMXBean
- LockServiceMXBean
- DiskStoreMXBean

- AsyncEventQueueMXBean
- LocatorMXBean
- LuceneServiceMXBean

JMX Manager nodes will have managed node MBeans for themselves since they are also manageable entities in the cluster.

MemberMXBean

Member's local view of its connection and cache. It is the primary gateway to manage a particular member. It exposes member level attributes and statistics. Some operations like `createCacheServer()` and `createManager()` will help to create some VMware GemFire resources. Any JMX client can connect to the MBean server and start managing a VMware GemFire Member by using this MBean.

See MemberMXBean Notifications for a list of notifications emitted by this MBean.

MBean Details

Scope	Local
Proxied	Yes
Object Name	GemFire:type=Member,member=<name-or-dist-member-id>
Instances Per Node	1

See the `org.apache.geode.management.MemberMXBean` JavaDocs for information on available MBean methods and attributes.

CacheServerMXBean

Represents the VMware GemFire CacheServer. Provides data and notifications about server, subscriptions, durable queues and indices.

See CacheServerMXBean Notifications for a list of notifications emitted by this MBean.

MBean Details

Scope	Local
Proxied	Yes
Object Name	GemFire:type=Member,service=CacheServer,member=<name-or-dist-member-id>
Instances Per Node	1

See the `org.apache.geode.management.CacheServerMXBean` JavaDocs for information on available MBean methods and attributes.

RegionMXBean

Member's local view of region.

MBean Details

Scope	Local
Proxied	Yes
Object Name	GemFire:type=Member,service=Region,name=<regionName>,member=<name-or-dist-member-id>
Instances Per Node	0..N

See the `org.apache.geode.management.RegionMXBean` JavaDocs for information on available MBean methods and attributes.

LockServiceMXBean

Represents a named instance of a LockService . Any number of LockServices can be created in a member.

MBean Details

Scope	Local
Proxied	Yes
Object Name	GemFire:type=Member,service=LockService,name=<dlsName>,member=<name-or-dist-member-id>
Instances Per Node	0..N

See the `org.apache.geode.management.LockServiceMXBean` JavaDocs for information on available MBean methods and attributes.

DiskStoreMXBean

Represents a DiskStore object which provides disk storage for one or more regions

MBean Details

Scope	Local
Proxied	Yes
Object Name	GemFire:type=Member,service=DiskStore,name=<name>,member=<name-or-dist-member-id>
Instances Per Node	0..N

See the `org.apache.geode.management.DiskStoreMXBean` JavaDocs for information on available MBean methods and attributes.

AsyncEventQueueMXBean

An AsyncEventQueueMXBean provides access to an AsyncEventQueue, which represent the channel over which events are delivered to the AsyncEventListener.

MBean Details

Scope	Local
Proxied	Yes
Object Name	GemFire:type=Member,service=AsyncEventQueue,queue=<queue-id>,member=<name-or-dist-member-id>
Instances Per Node	0..N

See the `org.apache.geode.management.AsyncEventQueueMXBean` JavaDocs for information on available MBean methods and attributes.

LocatorMXBean

A LocatorMXBean represents a locator.

MBean Details

Scope	Local
Proxied	Yes
Object Name	GemFire:type=Member,service=Locator,port=<port>,member=<name-or-dist-member-id>
Instances Per Node	0..1

See the `org.apache.geode.management.LocatorMXBean` JavaDocs for information on available MBean methods and attributes.

LuceneServiceMXBean

The member's local view of existing Lucene indexes.

MBean Details

Scope	Local
Proxied	Yes
Object Name	GemFire:service=CacheService,name=LuceneService,type=Member,member=<name-or-dist-member-id>
Instances Per Node	0..1

See the `org.apache.geode.cache.lucene.management.LuceneServiceMXBean` JavaDocs for information on available MBean methods and attributes.

GatewaySenderMXBean

A GatewaySenderMXBean represents a gateway sender.

MBean Details

Scope	Local
Proxied	Yes
Object Name	GemFire:type=Member,service=GatewaySender,gatewaySender=<sender-id>,member=<name-or-dist-member-id>
Instances Per Node	0..1

See the `org.apache.geode.management.GatewaySenderMXBean` JavaDocs for information on available MBean methods and attributes.

GatewayReceiverMXBean

A GatewayReceiverMXBean represents a gateway receiver.

MBean Details

Scope	Local
Proxied	Yes
Object Name	GemFire:type=Member,service=GatewayReceiver,member=<name-or-dist-member-id>
Instances Per Node	0..1

See the `org.apache.geode.management.GatewayReceiverMXBean` JavaDocs for information on available MBean methods and attributes.

Browsing VMware GemFire MBeans through JConsole

You can browse all the VMware GemFire MBeans in your cluster by using JConsole.

To view VMware GemFire MBeans through JConsole, perform the following steps:

1. Start a `gfsh` prompt.
2. Connect to a running cluster by either connecting to a locator with an embedded JMX Manager or connect directly to a JMX Manager. For example:

   ```
   gfsh>connect --locator=locator1[10334]
   ```

   or

   ```
   gfsh>connect --jmx-manager=locator1[1099]
   ```

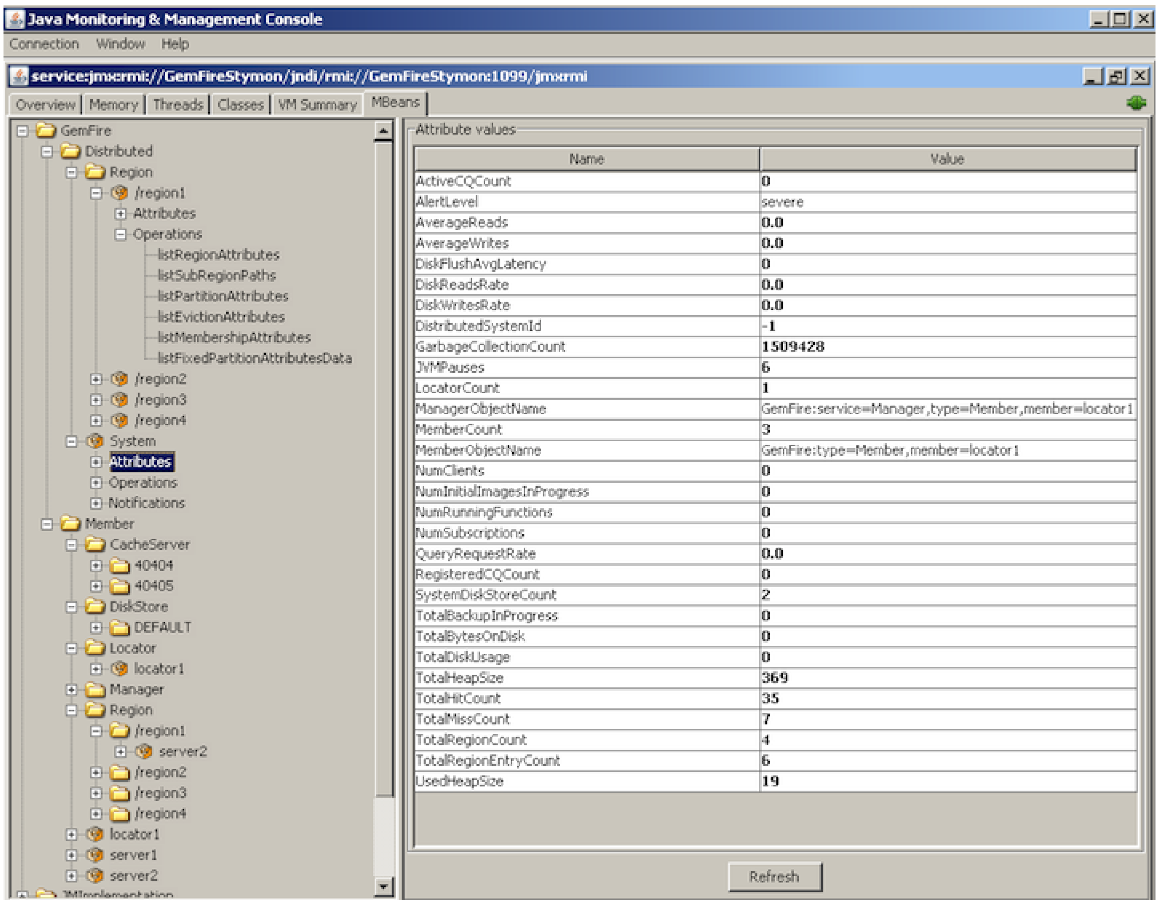
3. Start JConsole:

   ```
   gfsh>start jconsole
   ```

If successful, the message `Running JDK JConsole` appears. The JConsole application launches and connects directly to the JMX Manager using RMI.

4. On the JConsole screen, click on the MBeans tab. Expand **GemFire**. Then expand each MBean to browse individual MBean attributes, operations and notifications.

   The following is an example screenshot of the MBean hierarchy in a VMware GemFire cluster:



Attribute values

Name	Value
ActiveCQCount	0
AlertLevel	severe
AverageReads	0.0
AverageWrites	0.0
DiskFlushAvgLatency	0
DiskReadsRate	0.0
DiskWritesRate	0.0
DistributedSystemId	-1
GarbageCollectionCount	1509428
JVMPauses	6
LocatorCount	1
ManagerObjectName	GemFire:service=Manager,type=Member,member=locator1
MemberCount	3
MemberObjectName	GemFire:type=Member,member=locator1
NumClients	0
NumInitialImagesInProgress	0
NumRunningFunctions	0
NumSubscriptions	0
QueryRequestRate	0.0
RegisteredCQCount	0
SystemDiskStoreCount	2
TotalBackupInProgress	0
TotalBytesOnDisk	0
TotalDiskUsage	0
TotalHeapSize	369
TotalHitCount	35
TotalMissCount	7
TotalRegionCount	4
TotalRegionEntryCount	6
UsedHeapSize	19

VMware GemFire JMX MBean Notifications

VMware GemFire MBeans emit notifications when specific events occur or if an alert is raised in the VMware GemFire system. Using standard JMX APIs, users can add notification handlers to listen for these events.

- **Notification Federation**

  All notifications emitted from managed nodes are federated to all JMX Managers in the system.

- **List of JMX MBean Notifications**

  This topic lists all available JMX notifications emitted by VMware GemFire MBeans.

Notification Federation

All notifications emitted from managed nodes are federated to all JMX Managers in the system.

These notifications are federated and then emitted by the DistributedSystemMXBean. If you attach a `javax.management.NotificationListener` to your DistributedSystemMXBean, the NotificationListener can listen to notifications from all MemberMXBeans and all CacheServerMXBeans.

Attaching Listeners to MXBeans

When you attach a notification listener to the DistributedSystemMXBean, the DistributedSystemMXBean then acts as the notification hub for the entire cluster. You do not have to attach a listener to each individual member or cache server MBean in order to listen to all the notifications in the cluster.

The following is an example of attaching a NotificationListener to an MBean using the JMX MBeanServer API:

```
NotificationListener myListener = ...
ObjectName mbeanName = ...
MBeanServer.addNotificationListener(mbeanName, myListener, null, null);
```

JMX Managers will emit notifications for all cluster members with two exceptions:

- If you use cache.xml to define resources such as regions and disks, then notifications for these resources are not federated to the JMX Manager. In those cases, the DistributedSystemMXBean cannot emit those notifications.
- If a JMX Manager is started after a resource has been created, the JMX Manager cannot emit notifications for that resource.

System Alert Notifications

System alerts are VMware GemFire alerts wrapped within a JMX notification. The JMX Manager registers itself as an alert listener with each member of the system, and by default, it receives all messages logged with the SEVERE alert level by any node in the cluster. Consequently, the DistributedSystemMXBean will then emit notifications for these alerts on behalf of the DistributedSystem.

By default, the JMX Manager registers itself to send notifications only for SEVERE level alerts. To change the alert level that the JMX Manager will send notifications for, use the `DistributedMXBean.changeAlertLevel` method. Possible alert levels to set are WARNING, ERROR, SEVERE, and NONE. After changing the level, the JMX Manager will only emit that level of log message as notifications.

Notification objects include **type**, **source** and **message** attributes. System alerts also include the **userData** attribute. For system alerts, the notification object attributes correspond to the following:

- **type**: system.alert
- **source**: Distributed System ID
- **message**: alert message
- **userData**: name or ID of the member that raised the alert

List of JMX MBean Notifications

This topic lists all available JMX notifications emitted by VMware GemFire MBeans.

Notifications are emitted by the following MBeans:

- **MemberMXBean Notifications**
- **MemberMXBean Gateway Notifications**
- **CacheServerMXBean Notifications**
- **DistributedSystemMXBean Notifications**

MemberMXBean Notifications

Notification Type	Notification Source	Message
gemfire.distributedsystem.cache.region.created	Member name or ID	Region Created with Name <Region Name>
gemfire.distributedsystem.cache.region.closed	Member name or ID	Region Destroyed/Closed with Name <Region Name>
gemfire.distributedsystem.cache.disk.created	Member name or ID	DiskStore Created with Name <DiskStore Name>
gemfire.distributedsystem.cache.disk.closed	Member name or ID	DiskStore Destroyed/Closed with Name <DiskStore Name>
gemfire.distributedsystem.cache.lockservice.created	Member name or ID	LockService Created with Name <LockService Name>
gemfire.distributedsystem.cache.lockservice.closed	Member name or ID	Lockservice Closed with Name <LockService Name>
gemfire.distributedsystem.async.event.queue.created	Member name or ID	Async Event Queue is Created in the VM
gemfire.distributedsystem.cache.server.started	Member name or ID	Cache Server is Started in the VM
gemfire.distributedsystem.cache.server.stopped	Member name or ID	Cache Server is stopped in the VM
gemfire.distributedsystem.locator.started	Member name or ID	Locator is Started in the VM

MemberMXBean Gateway Notifications

Notification Type	Notification Source	Message
gemfire.distributedsystem.gateway.sender.created	Member name or ID	GatewaySender Created in the VM
gemfire.distributedsystem.gateway.sender.started	Member name or ID	GatewaySender Started in the VM <Sender Id>

Notification Type	Notification Source	Message
gemfire.distributedsystem.gateway.sender.stopped	Member name or ID	GatewaySender Stopped in the VM <Sender Id>
gemfire.distributedsystem.gateway.sender.paused	Member name or ID	GatewaySender Paused in the VM <Sender Id>
gemfire.distributedsystem.gateway.sender.resumed	Member name or ID	GatewaySender Resumed in the VM <Sender Id>
gemfire.distributedsystem.gateway.receiver.created	Member name or ID	GatewayReceiver Created in the VM
gemfire.distributedsystem.gateway.receiver.started	Member name or ID	GatewayReceiver Started in the VM
gemfire.distributedsystem.gateway.receiver.stopped	Member name or ID	GatewayReceiver Stopped in the VM
gemfire.distributedsystem.cache.server.started	Member name or ID	Cache Server is Started in the VM

CacheServerMXBean Notifications

Notification Type	Notification Source	Message
gemfire.distributedsystem.cacheserver.client.joined	CacheServer MBean Name	Client joined with Id <Client ID>
gemfire.distributedsystem.cacheserver.client.left	CacheServer MBean Name	Client crashed with Id <Client ID>
gemfire.distributedsystem.cacheserver.client.crashed	CacheServer MBean name	Client left with Id <Client ID>

DistributedSystemMXBean Notifications

Notification Type	Notification Source	Message
gemfire.distributedsystem.cache.member.joined	Name or ID of member who joined	Member Joined <Member Name or ID>
gemfire.distributedsystem.cache.member.departed	Name or ID of member who departed	Member Departed <Member Name or ID> has crashed = <true/false>
gemfire.distributedsystem.cache.member.suspect	Name or ID of member who is suspected	Member Suspected <Member Name or ID> By <Who Suspected>
system.alert.*	DistributedSystem("<Distributed System ID">)	Alert Message

Configuring RMI Registry Ports and RMI Connectors

VMware GemFire programmatically emulates out-of-the-box JMX provided by Java and creates a JMXServiceURL with RMI Registry and RMI Connector ports on all manageable members.

Configuring JMX Manager Port and Bind Addresses

You can configure a specific connection port and address when launching a process that will host the VMware GemFire JMX Manager. To do this, specify values for the `jmx-manager-bind-address`, which specifies the JMX manager's IP address and `jmx-manager-port`, which defines the RMI connection port.

The default VMware GemFire JMX Manager RMI port is 1099. You may need to modify this default if 1099 is reserved for other uses.

Using Out-of-the-Box RMI Connectors

If for some reason you need to use standard JMX RMI in your deployment for other monitoring purposes, set the VMware GemFire property `jmx-manager-port` to 0 on any members where you want to use standard JMX RMI.

If you use out-of-the-box JMX RMI instead of starting an embedded VMware GemFire JMX Manager, you should consider setting `-Dsun.rmi.dgc.server.gcInterval=Long.MAX_VALUE-1` when starting the JVM for customer applications and client processes. Every VMware GemFire process internally sets this setting before creating and starting the JMX RMI connector in order to prevent full garbage collection from pausing processes.

Executing gfsh Commands through the Management API

You can also use management APIs to execute gfsh commands programmatically.

Note: If you start the JMX Manager programmatically and wish to enable command processing, you must also add the absolute path of `gfsh-dependencies.jar` (located in `$GEMFIRE/lib` of your VMware GemFire installation) to the CLASSPATH of your application. Do not copy this library to your CLASSPATH because this library refers to other dependencies in `$GEMFIRE/lib` by a relative path. The following code samples demonstrate how to process and execute `gfsh` commands using the Java API.

First, retrieve a CommandService instance.

Note: The CommandService API is currently only available on JMX Manager nodes.

```
// Get existing CommandService instance or create new if it doesn't exist
commandService = CommandService.createLocalCommandService(cache);

// OR simply get CommandService instance if it exists, don't create new one
CommandService commandService = CommandService.getUsableLocalCommandService();
```

Next, process the command and its output:

```
// Process the user specified command String
Result regionListResult = commandService.processCommand("list regions");

// Iterate through Command Result in String form line by line
while (regionListResult.hasNextLine()) {
   System.out.println(regionListResult.nextLine());
}
```

Alternatively, instead of processing the command, you can create a CommandStatement Object from the command string which can be re-used.

```
// Create a command statement that can be reused multiple times
CommandStatement showDeadLocksCmdStmt = commandService.createCommandStatement
    ("show dead-locks --file=deadlock-info.txt");
Result showDeadlocksResult = showDeadLocksCmdStmt.process();

// If there is a file as a part of Command Result, it can be saved
// to a specified directory
if (showDeadlocksResult.hasIncomingFiles()) {
    showDeadlocksResult.saveIncomingFiles(System.getProperty("user.dir") +
                  "/commandresults");
}
```

Managing Heap and Off-heap Memory

By default, VMware GemFire uses the JVM heap. VMware GemFire also offers an option to store data off heap. This section describes how to manage heap and off-heap memory to best support your application.

Tuning the JVM’s Garbage Collection Parameters

Because VMware GemFire is specifically designed to manipulate data held in memory, you can optimize your application’s performance by tuning the way VMware GemFire uses the JVM heap.

See your JVM documentation for all JVM-specific settings that can be used to improve garbage collection (GC) response. At a minimum, do the following:

1. Set the initial and maximum heap switches, `-Xms` and `-Xmx`, to the same values. The `gfsh start server` options `--initial-heap` and `--max-heap` accomplish the same purpose, with the added value of providing resource manager defaults such as eviction threshold and critical threshold.
2. Configure your JVM for concurrent mark-sweep (CMS) garbage collection.
3. If your JVM allows, configure it to initiate CMS collection when heap use is at least 10% lower than your setting for the resource manager `eviction-heap-percentage`. You want the collector to be working when VMware GemFire is evicting or the evictions will not result in more free memory. For example, if the `eviction-heap-percentage` is set to 65, set your garbage collection to start when the heap use is no higher than 55%.

JVM	CMS switch flag	CMS initiation (begin at heap % N)
Sun HotSpot	`-XX:+UseConcMarkSweepGC`	`-XX:CMSInitiatingOccupancyFraction=N`
JRockit	`-Xgc:gencon`	`-XXgcTrigger:N`
IBM	`-Xgcpolicy:gencon`	N/A

For the `gfsh start server` command, pass these settings with the `--J` switch, for example: `--J=-XX:+UseConcMarkSweepGC`.

The following is an example of setting JVM for an application:

```
$ java app.MyApplication -Xms=30m -Xmx=30m -XX:+UseConcMarkSweepGC -XX:CMSInitiatingOc
cupancyFraction=60
```

Note: Do not use the `-XX:+UseCompressedStrings` and `-XX:+UseStringCache` JVM configuration properties when starting up servers. These JVM options can cause issues with data corruption and compatibility.

Or, using `gfsh`:

```
$ gfsh start server --name=app.MyApplication --initial-heap=30m --max-heap=30m \
--J=-XX:+UseConcMarkSweepGC --J=-XX:CMSInitiatingOccupancyFraction=60
```

Using the VMware GemFire Resource Manager

The VMware GemFire resource manager works with your JVM's tenured garbage collector to control heap use and protect your member from hangs and crashes due to memory overload.

The VMware GemFire resource manager prevents the cache from consuming too much memory by evicting old data. If the garbage collector is unable to keep up, the resource manager refuses additions to the cache until the collector has freed an adequate amount of memory.

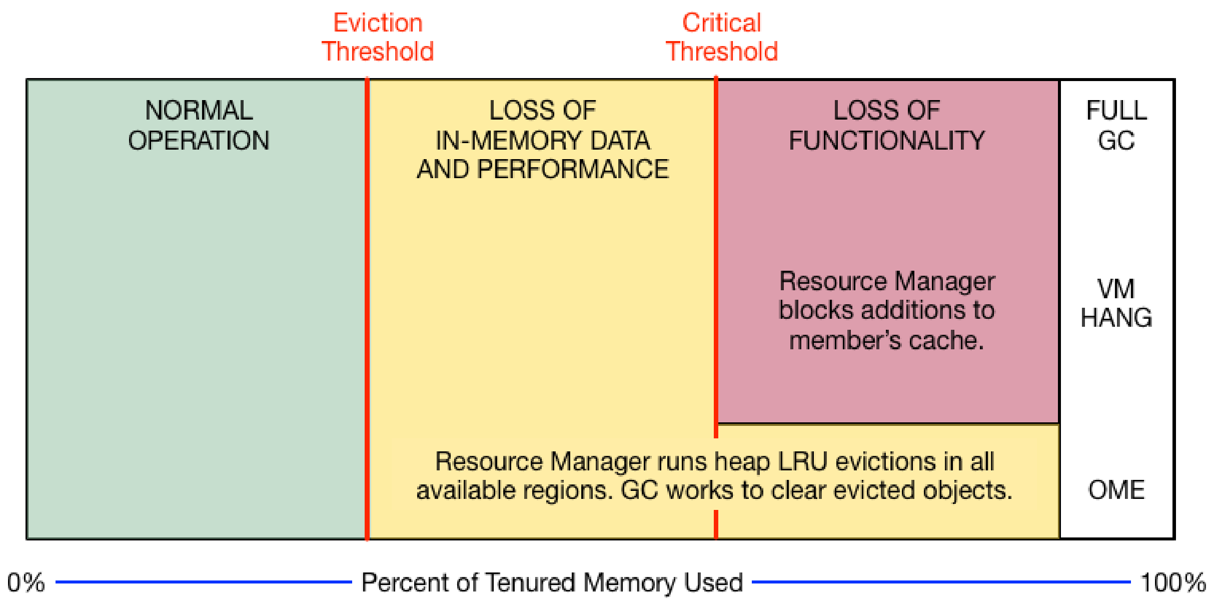
The resource manager has two threshold settings, each expressed as a percentage of the total tenured heap. Both are deactivated by default.

1. **Eviction Threshold**. Above this, the manager orders evictions for all regions with `eviction-attributes` set to `lru-heap-percentage`. This prompts dedicated background evictions, independent of any application threads and it also tells all application threads adding data to the regions to evict at least as much data as they add. The JVM garbage collector removes the evicted data, reducing heap use. The evictions continue until the manager determines that heap use is again below the eviction threshold.

   The resource manager enforces eviction thresholds only on regions whose LRU eviction policies are based on heap percentage. Regions whose eviction policies based on entry count or memory size use other mechanisms to manage evictions. See Eviction for more detail regarding eviction policies.

2. **Critical Threshold**. Above this, all activity that might add data to the cache is refused. This threshold is set above the eviction threshold and is intended to allow the eviction and GC work to catch up. This JVM, all other JVMs in the distributed system, and all clients to the system receive `LowMemoryException` for operations that would add to this critical member's heap consumption. Activities that fetch or reduce data are allowed. For a list of refused operations, see the Javadocs for the `ResourceManager` method `setCriticalHeapPercentage`.

   Critical threshold is enforced on all regions, regardless of LRU eviction policy, though it can be set to zero to deactivate its effect.


Eviction Threshold
Critical Threshold
NORMAL OPERATION
LOSS OF IN-MEMORY DATA AND PERFORMANCE
LOSS OF FUNCTIONALITY
FULL GC
Resource Manager blocks additions to member's cache.
VM HANG
Resource Manager runs heap LRU evictions in all available regions. GC works to clear evicted objects.
OME
0%
Percent of Tenured Memory Used
100%

When heap use passes the eviction threshold in either direction, the manager logs an info-level message.

When heap use exceeds the critical threshold, the manager logs an error-level message. Avoid exceeding the critical threshold. Once identified as critical, the VMware GemFire member becomes a read-only member that refuses cache updates for all of its regions, including incoming distributed updates.

For more information, see `org.apache.geode.cache.control.ResourceManager` in the online API documentation.

How Background Eviction Is Performed

When the manager kicks off evictions:

1. From all regions in the local cache that are configured for heap LRU eviction, the background eviction manager creates a randomized list containing one entry for each partitioned region bucket (primary or secondary) and one entry for each non-partitioned region. So each partitioned region bucket is treated the same as a single, non-partitioned region.
2. The background eviction manager starts four evictor threads for each processor on the local machine. The manager passes each thread its share of the bucket/region list. The manager divides the bucket/region list as evenly as possible by count, and not by memory consumption.
3. Each thread iterates round-robin over its bucket/region list, evicting one LRU entry per bucket/region until the resource manager sends a signal to stop evicting.

See also Memory Requirements for Cached Data.

Controlling Heap Use with the Resource Manager

Resource manager behavior is closely tied to the triggering of Garbage Collection (GC) activities, the use of concurrent garbage collectors in the JVM, and the number of parallel GC threads used for concurrency.

The recommendations provided here for using the manager assume you have a solid understanding of your Java VM's heap management and garbage collection service.

The resource manager is available for use in any VMware GemFire member, but you may not want to activate it everywhere. For some members it might be better to occasionally restart after a hang or OME crash than to evict data and/or refuse distributed caching activities. Also, members that do not risk running past their memory limits would not benefit from the overhead the resource manager consumes. Cache servers are often configured to use the manager because they generally host more data and have more data activity than other members, requiring greater responsiveness in data cleanup and collection.

For the members where you want to activate the resource manager:

1. Configure VMware GemFire for heap LRU management.
2. Set the JVM GC tuning parameters to handle heap and garbage collection in conjunction with the VMware GemFire manager.
3. Monitor and tune heap LRU configurations and your GC configurations.
4. Before going into production, run your system tests with application behavior and data loads that approximate your target systems so you can tune as well as possible for production needs.
5. In production, keep monitoring and tuning to meet changing needs.

Configure VMware GemFire for Heap LRU Management

The configuration terms used here are `cache.xml` elements and attributes, but you can also configure through `gfsh` and the `org.apache.geode.cache.control.ResourceManager` and `Region` APIs.

1. When starting up your server, set `initial-heap` and `max-heap` to the same value.
2. Set the `resource-manager` `critical-heap-percentage` threshold. This should be as as close to 100 as possible while still low enough so the manager's response can prevent the member from hanging or getting `OutOfMemoryError`. The threshold is zero (no threshold) by default. **Note:** When you set this threshold, it also enables a query monitoring feature that prevents most out-of-memory exceptions when executing queries or creating indexes. See Monitoring Queries for Low Memory.
3. Set the `resource-manager` `eviction-heap-percentage` threshold to a value lower than the critical threshold. This should be as high as possible while still low enough to prevent your member from reaching the critical threshold. The threshold is zero (no threshold) by default.
4. Decide which regions will participate in heap eviction and set their `eviction-attributes` to `lru-heap-percentage`. See Eviction. The regions you configure for eviction should have enough data activity for the evictions to be useful and should contain data your application can afford to delete or offload to disk.

gfsh example:

```
gfsh>start server --name=server1 --initial-heap=30m --max-heap=30m \
--critical-heap-percentage=80 --eviction-heap-percentage=60
```

cache.xml example:

```
<cache>
<region refid="REPLICATE_HEAP_LRU" />
...
<resource-manager critical-heap-percentage="80" eviction-heap-percentage="60"/>
</cache>
```

Note: The `resource-manager` specification must appear after the region declarations in your cache.xml file.

Set the JVM GC Tuning Parameters

If your JVM allows, configure it to initiate concurrent mark-sweep (CMS) garbage collection when heap use is at least 10% lower than your setting for the resource manager `eviction-heap-percentage`. You want the collector to be working when VMware GemFire is evicting or the evictions will not result in more free memory. For example, if the `eviction-heap-percentage` is set to 65, set your garbage collection to start when the heap use is no higher than 55%.

Monitor and Tune Heap LRU Configurations

In tuning the resource manager, your central focus should be keeping the member below the critical threshold. The critical threshold is provided to avoid member hangs and crashes, but because of its exception-throwing behavior for distributed updates, the time spent in critical negatively impacts the entire distributed system. To stay below critical, tune so that the VMware GemFire eviction and the JVM's GC respond adequately when the eviction threshold is reached.

Use the statistics provided by your JVM to make sure your memory and GC settings are sufficient for your needs.

The VMware GemFire `ResourceManagerStats` provide information about memory use and the manager thresholds and eviction activities.

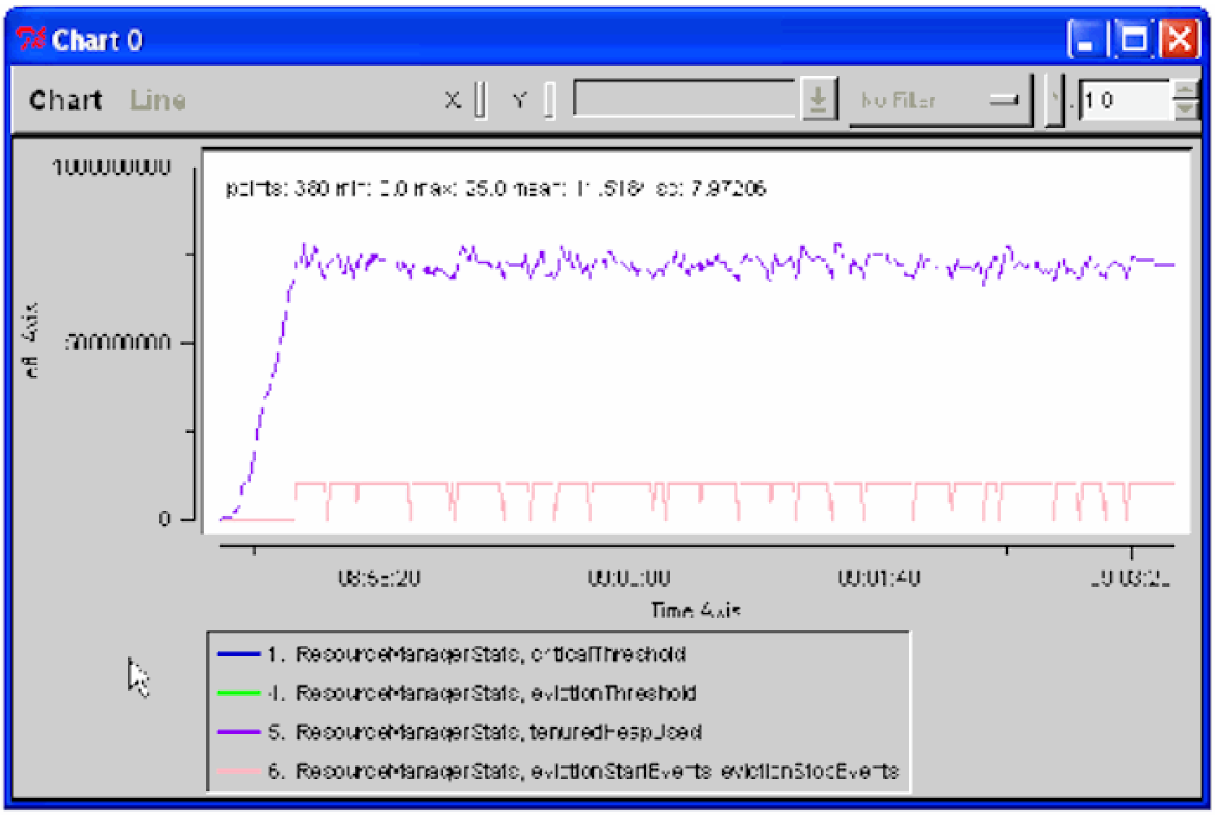
If your application spikes above the critical threshold on a regular basis, try lowering the eviction threshold. If the application never goes near critical, you might raise the eviction threshold to gain more usable memory without the overhead of unneeded evictions or GC cycles.

The settings that will work well for your system depend on a number of factors, including these:

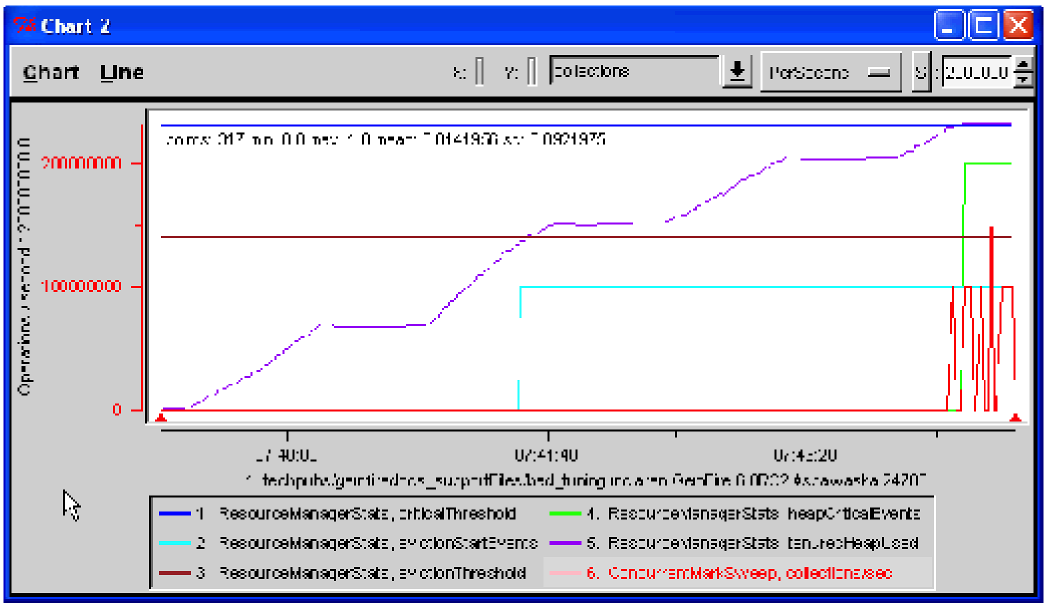
- **The size of the data objects you store in the cache:** Very large data objects can be evicted and garbage collected relatively quickly. The same amount of space in use by many small objects takes more processing effort to clear and might require lower thresholds to allow eviction and GC activities to keep up.
- **Application behavior:** Applications that quickly put a lot of data into the cache can more easily overrun the eviction and GC capabilities. Applications that operate more slowly may be more easily offset by eviction and GC efforts, possibly allowing you to set your thresholds higher than in the more volatile system.

- **Your choice of JVM:** Each JVM has its own GC behavior, which affects how efficiently the collector can operate, how quickly it kicks in when needed, and other factors.

In this sample statistics chart in VSD, the manager's evictions and the JVM's GC efforts are good enough to keep heap use very close to the eviction threshold. The eviction threshold could be increased to a setting closer to the critical threshold, allowing the member to keep more data in tenured memory without the risk of overwhelming the JVM. This chart also shows the blocks of times when the manager was running cache evictions.



In this next chart, it looks like the manager's evictions are kicking in at the right time, but the CMS garbage collector is not starting soon enough to keep memory use in check. It might be that it is not configured to start as soon as it should. It should be started just before the eviction threshold is reached. Or there might be some other issue with the garbage collection service.



Resource Manager Example Configurations

These examples set the critical threshold to 85 percent of the tenured heap and the eviction threshold to 75 percent. The region `bigDataStore` is configured to participate in the resource manager's eviction activities.

- gfsh Example:

```
gfsh>start server --name=server1 --initial-heap=30m --max-heap=30m \
--critical-heap-percentage=85 --eviction-heap-percentage=75
```

```
gfsh>create region --name=bigDataStore --type=PARTITION_HEAP_LRU
```

- XML:

```
<cache>
<region name="bigDataStore" refid="PARTITION_HEAP_LRU"/>
...
<resource-manager critical-heap-percentage="85" eviction-heap-percentage="75"/>
</cache>
```

Note: The `resource-manager` specification must appear after the region declarations in your cache.xml file.

- Java:

```
Cache cache = CacheFactory.create();

ResourceManager rm = cache.getResourceManager();
rm.setCriticalHeapPercentage(85);
rm.setEvictionHeapPercentage(75);

RegionFactory rf =
  cache.createRegionFactory(RegionShortcut.PARTITION_HEAP_LRU);
  Region region = rf.create("bigDataStore");
```

Use Case for the Example Code

This is one possible scenario for the configuration used in the examples:

- A 64-bit Java VM with 8 Gb of heap space on a 4 CPU system running Linux.
- The data region bigDataStore has approximately 2-3 million small values with average entry size of 512 bytes. So approximately 4-6 Gb of the heap is for region storage.
- The member hosting the region also runs an application that may take up to 1 Gb of the heap.
- The application must never run out of heap space and has been crafted such that data loss in the region is acceptable if the heap space becomes limited due to application issues, so the default `lru-heap-percentage` action destroy is suitable.
- The application's service guarantee makes it very intolerant of `OutOfMemoryException` errors. Testing has shown that leaving 15% head room above the critical threshold when adding data to the region gives 99.5% uptime with no `OutOfMemoryException` errors, when configured with the CMS garbage collector using `-XX:CMSInitiatingOccupancyFraction=70`.

Managing Off-Heap Memory

VMware GemFire can be configured to store region values in off-heap memory, which is memory within the JVM that is not subject to Java garbage collection.

Garbage collection (GC) within a JVM can prove to be a performance impediment. A server cannot exert control over when garbage collection within the JVM heap memory takes place, and the server has little control over the triggers for invocation. Off-heap memory offloads values to a storage area that is not subject to Java GC. By taking advantage of off-heap storage, an application can reduce the amount of heap storage that is subject to GC overhead.

Off-heap memory works in conjunction with the heap, it does not replace it. The keys are stored in heap memory space. VMware GemFire's own memory manager handles the off-heap memory with better performance than the Java garbage collector would for certain sets of region data.

The resource manager monitors the contents of off-heap memory and invokes memory management operations in accordance with two thresholds similar to those used for monitoring the JVM heap: `eviction-off-heap-percentage` and `critical-off-heap-percentage`.

On-heap and Off-heap Objects

The following objects are always stored in the JVM heap:

- Region metadata
- Entry metadata
- Keys
- Indexes
- Subscription queue elements

The following objects can be stored in off-heap memory:

- Values - maximum value size is 2GB

- Reference counts
- List of free memory blocks
- WAN queue elements

Note: Do not use functional range indexes with off-heap data, as they are not supported. An attempt to do so generates an exception.

Off-heap Recommendations

Off-heap storage is best suited to data patterns where:

- Stored values are relatively uniform in size
- Stored values are mostly less than 128K in size
- The usage patterns involve cycles of many creates followed by destroys or clear
- The values do not need to be frequently deserialized
- Many of the values are long-lived reference data

Be aware that VMware GemFire has to perform extra work to access the data stored in off-heap memory since it is stored in serialized form. This extra work may cause some use cases to run slower in an off-heap configuration, even though they use less memory and avoid garbage collection overhead. However, even with the extra deserialization, off-heap storage may give you the best performance. Features that may increase overhead include

- frequent updates
- stored values of widely varying sizes
- deltas
- queries

Implementation Details

The off-heap memory manager is efficient at handling region data values that are all the same size or are of fixed sizes. With fixed and same-sized data values allocated within the off-heap memory, freed chunks can often be re-used, and there is little or no need to devote cycles to defragmentation.

Region values that are less than or equal to eight bytes in size will not reside in off-heap memory, even if the region is configured to use off-heap memory. These very small size region values reside in the JVM heap in place of a reference to an off-heap location. This performance enhancement saves space and load time.

Controlling Off-heap Use with the Resource Manager

The VMware GemFire resource manager controls off-heap memory by means of two thresholds, in much the same way as it does JVM heap memory. See Using the VMware GemFire Resource Manager. The resource manager prevents the cache from consuming too much off-heap memory by evicting old data. If the off-heap memory manager is unable to keep up, the resource manager refuses additions to the cache until the off-heap memory manager has freed an adequate amount of memory.

The resource manager has two threshold settings, each expressed as a percentage of the total off-heap memory. Both are disabled by default.

1. **Eviction Threshold**. The percentage of off-heap memory at which eviction should begin. Evictions continue until the resource manager determines that off-heap memory use is again below the eviction threshold. Set the eviction threshold with the `eviction-off-heap-percentage` region attribute. The resource manager enforces an eviction threshold only on regions with the HEAP_LRU characteristic. If critical threshold is non-zero, the default eviction threshold is 5% below the critical threshold. If critical threshold is zero, the default eviction threshold is 80% of total off-heap memory.

   The resource manager enforces eviction thresholds only on regions whose LRU eviction policies are based on heap percentage. Regions whose eviction policies based on entry count or memory size use other mechanisms to manage evictions. See Eviction for more detail regarding eviction policies.

2. **Critical Threshold**. The percentage of off-heap memory at which the cache is at risk of becoming inoperable. When cache use exceeds the critical threshold, all activity that might add data to the cache is refused. Any operation that would increase consumption of off-heap memory throws a `LowMemoryException` instead of completing its operation. Set the critical threshold with the `critical-off-heap-percentage` region attribute.

   Critical threshold is enforced on all regions, regardless of LRU eviction policy, though it can be set to zero to disable its effect.

Specifying Off-heap Memory

To use off-heap memory, specify the following options when setting up servers and regions:

- Start the JVM as described in Tuning the JVM's Garbage Collection Parameters. In particular, set the initial and maximum heap sizes to the same value. Sizes less than 32GB are optimal when you plan to use off-heap memory.
- From gfsh, start each server that will support off-heap memory with a non-zero `off-heap-memory-size` value, specified in megabytes (m) or gigabytes (g). If you plan to use the resource manager, specify critical threshold, eviction threshold, or (in most cases) both.

  Example:

  ```
  gfsh> start server --name=server1 --initial-heap=10G --max-heap=10G --off-heap-
  memory-size=200G \
  --lock-memory=true --critical-off-heap-percentage=90 --eviction-off-heap-percen
  tage=80
  ```

- Mark regions whose entry values should be stored off-heap by setting the `off-heap` region attribute to `true` Configure other region attributes uniformly for all members that host data for the same region. .

  Example:

  ```
  gfsh>create region --name=region1 --type=PARTITION_HEAP_LRU --off-heap=true
  ```

gfsh Off-heap Support

gfsh supports off-heap memory in server and region creation operations and in reporting functions:

alter disk-store
`--off-heap=(true | false)` resets the off-heap attribute for the specified region. See alter disk-store for details.

create region
`--off-heap=(true | false)` sets the off-heap attribute for the specified region. See create region for details.

describe member
displays off-heap size

describe offline-disk-store
shows if an off-line region is off-heap

describe region
displays the value of a region's off-heap attribute

show metrics
includes off-heap metrics `maxMemory`, `freeMemory`, `usedMemory`, `objects`, `fragmentation` and `defragmentationTime`

start server
supports off-heap options `--lock-memory`, `--off-heap-memory-size`, `--critical-off-heap-percentage`, and `--eviction-off-heap-percentage` See start server for details.

ResourceManager API

The `org.apache.geode.cache.control.ResourceManager` interface defines methods that support off-heap use:

- `public void setCriticalOffHeapPercentage(float Percentage)`
- `public float getCriticalOffHeapPercentage()`
- `public void setEvictionOffHeapPercentage(float Percentage)`
- `public float getEvictionOffHeapPercentage()`

The gemfire.properties file supports one off-heap property:

`off-heap-memory-size`
Specifies the size of off-heap memory in megabytes (m) or gigabytes (g). For example:

```
off-heap-memory-size=4096m
off-heap-memory-size=120g
```

See gemfire.properties and gfsecurity.properties (VMware GemFire Properties) for details.

The cache.xml file supports one region attribute:

`off-heap(=true | false)`
Specifies that the region uses off-heap memory; defaults to `false`. For example:

```
<region-attributes
  off-heap="true">
</region-attributes>
```

See <region-attributes> for details.

The cache.xml file supports two resource manager attributes:

`critical-off-heap-percentage=value`
Specifies the percentage of off-heap memory at or above which the cache is considered in danger of becoming inoperable due to out of memory exceptions. See <resource-manager> for details.

`eviction-off-heap-percentage=value`
Specifies the percentage of off-heap memory at or above which eviction should begin. Can be set for any region, but actively operates only in regions configured for HEAP_LRU eviction. See <resource-manager> for details.

For example:

```
<cache>
...
   <resource-manager
      critical-off-heap-percentage="99.9"
      eviction-off-heap=-percentage="85"/>
...
</cache>
```

Tuning Off-heap Memory Usage

VMware GemFire collects statistics on off-heap memory usage which you can view with the gfsh `show metrics` command. See Off-Heap (OffHeapMemoryStats) for a description of available off-heap statistics.

Off-heap memory is optimized, by default, for storing values of 128 KB in size. This figure is known as the “maximum optimized stored value size,” which we will denote here by *maxOptStoredValSize*. If your data typically runs larger, you can enhance performance by increasing the OFF_HEAP_FREE_LIST_COUNT system parameter to a number larger than `maxOptStoredValSize/8`, where *maxOptStoredValSize* is expressed in KB (1024 bytes). So, the default values correspond to:

```
128 KB / 8 = (128 * 1024) / 8 = 131,072 / 8 = 16,384
-Dgemfire.OFF_HEAP_FREE_LIST_COUNT=16384
```

To optimize for a maximum optimized stored value size that is twice the default, or 256 KB, the free list count should be doubled:

```
-Dgemfire.OFF_HEAP_FREE_LIST_COUNT=32768
```

During the tuning process, you can toggle the `off-heap` region attribute on and off, leaving other off-heap settings and parameters in place, in order to compare your application’s on-heap and off-heap performance.

Locking Memory (Linux Systems Only)

On Linux systems, you can lock memory to prevent the operating system from paging out heap or off-heap memory.

To use this feature:

1. Configure the operating system limits for locked memory. Increase the operating system's `ulimit -l` value (the maximum size that may be locked in memory) from the default (typically 32 KB or 64 KB) to at least the total amount of memory used by VMware GemFire for on-heap or off-heap storage. To view the current setting, enter `ulimit -a` at a shell prompt and find the value for `max locked memory`:

   ```
   # ulimit -a
   ...
   max locked memory       (kbytes, -l) 64
   ...
   ```

   Use `ulimit -l max-size-in-kbytes` to raise the limit. For example, to set the locked memory limit to 64 GB:

   ```
   # ulimit -l 64000000
   ```

2. Using locked memory in this manner increases the time required to start VMware GemFire. The additional time required to start VMware GemFire depends on the total amount of memory used, and can range from several seconds to 10 minutes or more. To improve startup time and reduce the potential of member timeouts, instruct the kernel to free operating system page caches just before starting a VMware GemFire member by issuing the following command:

   ```
   $ echo 1 > /proc/sys/vm/drop_caches
   ```

3. Start each VMware GemFire data store with the gfsh `-lock-memory=true` option. If you deploy more than one server per host, begin by starting each server sequentially. Starting servers sequentially avoids a race condition in the operating system that can cause failures (even machine crashes) if you accidentally over-allocate the available RAM. After you verify that the system configuration is stable, you can then start servers concurrently.

Disk Storage

With VMware GemFire disk stores, you can persist data to disk as a backup to your in-memory copy and overflow data to disk when memory use gets too high.

- **How Disk Stores Work**

  Overflow and persistence use disk stores individually or together to store data.

- **Disk Store File Names and Extensions**

  Disk store files include store management files, access control files, and the operation log, or oplog, files, consisting of one file for deletions and another for all other operations.

- **Disk Store Operation Logs**

At creation, each operation log is initialized at the disk store's `max-oplog-size`, with the size divided between the `crf` and `drf` files. When the oplog is closed, VMware GemFire shrinks the files to the space used in each file.

- **Configuring Disk Stores**

  In addition to the disk stores you specify, VMware GemFire has a default disk store that it uses when disk use is configured with no disk store name specified. You can modify default disk store behavior.

- **Optimizing a System with Disk Stores**

  Optimize availability and performance by following the guidelines in this section.

- **Start Up and Shut Down with Disk Stores**

  This section describes what happens during startup and shutdown and provides procedures for those operations.

- **Disk Store Management**

  The `gfsh` command-line tool has a number of options for examining and managing your disk stores. The `gfsh` tool, the `cache.xml` file and the DiskStore APIs are your management tools for online and offline disk stores.

- **Creating Backups for System Recovery and Operational Management**

  A backup is a copy of persisted data from a disk store. A backup is used to restore the disk store to the state it was in when the backup was made. The appropriate back up and restore procedures differ based upon whether the cluster is online or offline. An online system has currently running members. An offline system does not have any running members.

How Disk Stores Work

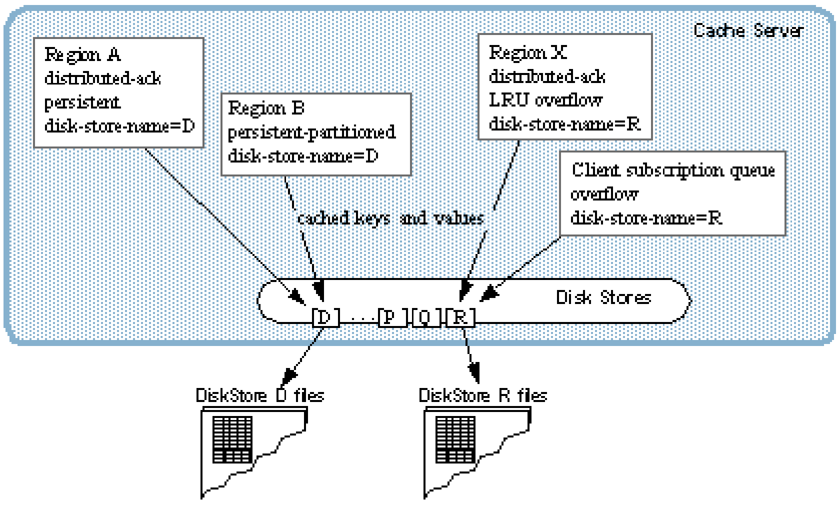
Overflow and persistence use disk stores individually or together to store data.

Disk storage is available for these items:

- **Regions**. Persist and/or overflow data from regions.
- **Server's client subscription queues**. Overflow the messaging queues to control memory use.
- **Gateway sender queues**. Persist these for high availability. These queues always overflow.
- **PDX serialization metadata**. Persist metadata about objects you serialize using VMware GemFire PDX serialization.

Each member has its own set of disk stores, and they are completely separate from the disk stores of any other member. For each disk store, define where and how the data is stored to disk. You can store data from multiple regions and queues in a single disk store.

This figure shows a member with disk stores D through R defined. The member has two persistent regions using disk store D and an overflow region and an overflow queue using disk store R.



What VMware GemFire Writes to the Disk Store

VMware GemFire writes the following to the disk store:

- Persisted and overflowed data as specified when the disk store was created and configured
- The members that host the store and information on their status, such as which members are online and which members are offline and time stamps
- A disk store identifier
- Which regions are in the disk store, specified by region name and including selected attributes
- Names of colocated regions on which the regions in the disk store depend
- A record of all operations on the regions

VMware GemFire does not write indexes to disk.

Disk Store State

The files for a disk store are used by VMware GemFire as a group. Treat them as a single entity. If you copy them, copy them all together. Do not change the file names.

Disk store access and management differs according to whether the member is online or offline. While a member is running, its disk stores are online. When the member exits and is not running, its disk stores are offline.

- Online, a disk store is owned and managed by its member process. To run operations on an online disk store, use API calls in the member process, or use the `gfsh` command-line interface.
- Offline, the disk store is just a collection of files in the host file system. The files are accessible based on file system permissions. You can copy the files for backup or to move the member's disk store location. You can also run some maintenance operations, such as file compaction and validation, by using the `gfsh` command-line interface. When offline, the disk store's information is unavailable to the cluster. For partitioned regions, region data is split between multiple members, and therefore the start up of a member is dependent on

all members, and must wait for all members to be online. An attempt to access an entry that is stored on disk by an offline member results in a `PartitionOfflineException`.

Disk Store File Names and Extensions

Disk store files include store management files, access control files, and the operation log, or oplog, files, consisting of one file for deletions and another for all other operations.

The next tables describe file names and extensions; they are followed by example disk store files.

File Names

File names have three parts: usage identifier, disk store name, and oplog sequence number.

First Part of File Name: Usage Identifier

Values	Used for	Examples
OVERFLOW	Oplog data from overflow regions and queues only.	OVERFLOWoverflowDS1_1.crf
BACKUP	Oplog data from persistent and persistent+overflow regions and queues.	BACKUPoverflowDS1.if, BACKUPDEFAULT.if
DRLK_IF	Access control - locking the disk store.	DRLK_IFoverflowDS1.lk, DRLK_IFDEFAULT.lk

Second Part of File Name: Disk Store Name

Values	Used for	Examples
<disk store name>	Non-default disk stores.	name="overflowDS1" DRLK_IFoverflowDS1.lk, name="persistDS1" BACKUPpersistDS1_1.crf
DEFAULT	Default disk store name, used when persistence or overflow are specified on a region or queue but no disk store is named.	DRLK_IFDEFAULT.lk, BACKUPDEFAULT_1.crf

Third Part of File Name: oplog Sequence Number

Values	Used for	Examples
Sequence number in the format _n	Oplog data files only. Numbering starts with 1.	OVERFLOWoverflowDS1_1.crf, BACKUPpersistDS1_2.crf, BACKUPpersistDS1_3.crf

File Extensions

File extension	Used for	Notes
if	Disk store metadata	Stored in the first disk-dir listed for the store. Negligible size - not considered in size control.
lk	Disk store access control	Stored in the first disk-dir listed for the store. Negligible size - not considered in size control.

File extension	Used for	Notes
crf	Oplog: create, update, and invalidate operations	Pre-allocated 90% of the total max-oplog-size at creation.
drf	Oplog: delete operations	Pre-allocated 10% of the total max-oplog-size at creation.
krf	Oplog: key and crf offset information	Created after the oplog has reached the max-oplog-size. Used to improve performance at startup.

Example files for disk stores persistDS1 and overflowDS1:

```
bash-2.05$ ls -tlr persistData1/
total 8
-rw-rw-r--   1 person users        188 Mar  4 06:17 BACKUPpersistDS1.if
-rw-rw-r--   1 person users          0 Mar  4 06:18 BACKUPpersistDS1_1.drf
-rw-rw-r--   1 person users         38 Mar  4 06:18 BACKUPpersistDS1_1.crf

bash-2.05$ ls -tlr overflowData1/
total 1028
-rw-rw-r--   1 person users          0 Mar  4 06:21 DRLK_IFoverflowDS1.lk
-rw-rw-r--   1 person users          0 Mar  4 06:21 BACKUPoverflowDS1.if
-rw-rw-r--   1 person users 1073741824 Mar  4 06:21 OVERFLOWoverflowDS1_1.crf
```

Example default disk store files for a persistent region:

```
bash-2.05$ ls -tlr
total 106
-rw-rw-r--   1 person users       1010 Mar  8 15:01 defTest.xml
drwxrwxr-x   2 person users        512 Mar  8 15:01 backupDirectory
-rw-rw-r--   1 person users          0 Mar  8 15:01 DRLK_IFDEFAULT.lk
-rw-rw-r--   1 person users  107374183 Mar  8 15:01 BACKUPDEFAULT_1.drf
-rw-rw-r--   1 person users  966367641 Mar  8 15:01 BACKUPDEFAULT_1.crf
-rw-rw-r--   1 person users        172 Mar  8 15:01 BACKUPDEFAULT.if
```

Disk Store Operation Logs

At creation, each operation log is initialized at the disk store's `max-oplog-size`, with the size divided between the `crf` and `drf` files. When the oplog is closed, VMware GemFire shrinks the files to the space used in each file.

After the oplog is closed, VMware GemFire also attempts to create a `krf` file, which contains the key names as well as the offset for the value within the `crf` file. Although this file is not required for startup, if it is available, it will improve startup performance by allowing VMware GemFire to load the entry values in the background after the entry keys are loaded.

When an operation log is full, VMware GemFire automatically closes it and creates a new log with the next sequence number. This is called *oplog rolling*. You can also request an oplog rolling through the API call `DiskStore.forceRoll`. You may want to do this immediately before compacting your disk stores, so the latest oplog is available for compaction.

Note: Log compaction can change the names of the disk store files. File number sequencing is usually altered, with some existing logs removed or replaced by newer logs with higher numbering. VMware GemFire always starts a new log at a number higher than any existing number.

This example listing shows the logs in a system with only one disk directory specified for the store. The first log (`BACKUPCacheOverflow_1.crf` and `BACKUPCacheOverflow_1.drf`) has been closed and the system is writing to the second log.

```
bash-2.05$ ls -tlra
total 55180
drwxrwxr-x   7 person users        512 Mar 22 13:56 ..
-rw-rw-r--   1 person users          0 Mar 22 13:57 BACKUPCacheOverflow_2.drf
-rw-rw-r--   1 person users     426549 Mar 22 13:57 BACKUPCacheOverflow_2.crf
-rw-rw-r--   1 person users          0 Mar 22 13:57 BACKUPCacheOverflow_1.drf
-rw-rw-r--   1 person users     936558 Mar 22 13:57 BACKUPCacheOverflow_1.crf
-rw-rw-r--   1 person users       1924 Mar 22 13:57 BACKUPCacheOverflow.if
drwxrwxr-x   2 person users       2560 Mar 22 13:57 .
```

The system rotates through all available disk directories to write its logs. The next log is always started in a directory that has not reached its configured capacity, if one exists.

When Disk Store Oplogs Reach the Configured Disk Capacity

If no directory exists that is within its capacity limits, how VMware GemFire handles this depends on whether automatic compaction is enabled.

- If auto-compaction is enabled, VMware GemFire creates a new oplog in one of the directories, going over the limit, and logs a warning that reports:

  ```
  Even though the configured directory size limit has been exceeded a
  new oplog will be created. The current limit is of XXX. The current
  space used in the directory is YYY.
  ```

  Note: When auto-compaction is enabled, `dir-size` does not limit how much disk space is used. VMware GemFire will perform auto-compaction, which should free space, but the system may go over the configured disk limits.

- If auto-compaction is disabled, VMware GemFire does not create a new oplog, operations in the regions attached to the disk store block, and VMware GemFire logs this error:

  ```
  Disk is full and rolling is disabled. No space can be created
  ```

Configuring Disk Stores

In addition to the disk stores you specify, VMware GemFire has a default disk store that it uses when disk use is configured with no disk store name specified. You can modify default disk store behavior.

- **Designing and Configuring Disk Stores**

  You define disk stores in your cache, then you assign them to your regions and queues by setting the `disk-store-name` attribute in your region and queue configurations.

- **Disk Store Configuration Parameters**

You define your disk stores by using the `gfsh create disk-store` command or in `<disk-store>` subelements of your cache declaration in `cache.xml`. All disk stores are available for use by all of your regions and queues.

- **Modifying the Default Disk Store**

  You can modify the behavior of the default disk store by specifying the attributes you want for the disk store named “DEFAULT”.

Designing and Configuring Disk Stores

You define disk stores in your cache, then you assign them to your regions and queues by setting the `disk-store-name` attribute in your region and queue configurations.

Note: Besides the disk stores you specify, VMware GemFire has a default disk store that it uses when disk use is configured with no disk store name specified. By default, this disk store is saved to the application’s working directory. You can change its behavior, as indicated in Create and Configure Your Disk Stores and Modifying the Default Disk Store.

- Design Your Disk Stores
- Create and Configure Your Disk Stores
- Configuring Regions, Queues, and PDX Serialization to Use the Disk Stores
- Configuring Disk Stores on Gateway Senders

Design Your Disk Stores

Before you begin, you should understand VMware GemFire Basic Configuration and Programming.

1. Work with your system designers and developers to plan for anticipated disk storage requirements in your testing and production caching systems. Take into account space and functional requirements.

   - For efficiency, separate data that is only overflowed in separate disk stores from data that is persisted or persisted and overflowed. Regions can be overflowed, persisted, or both. Server subscription queues are only overflowed.
   - When calculating your disk requirements, figure in your data modification patterns and your compaction strategy. VMware GemFire creates each oplog file at the max-oplog-size, which defaults to 1 GB. Obsolete operations are removed from the oplogs only during compaction, so you need enough space to store all operations that are done between compactions. For regions where you are doing a mix of updates and deletes, if you use automatic compaction, a good upper bound for the required disk space is

     ```
     (1 / (compaction_threshold/100) ) * data size
     ```

     where data size is the total size of all the data you store in the disk store. So, for the default compaction-threshold of 50, the disk space is roughly twice your data size. Note that the compaction thread could lag behind other operations, causing disk use to rise temporarily above the upper bound. If you disable automatic compaction, the amount of disk required depends on how many obsolete operations accumulate between manual compactions.

2. Work with your host system administrators to determine where to place your disk store directories, based on your anticipated disk storage requirements and the available disks on your host systems.
    - Make sure the new storage does not interfere with other processes that use disk on your systems. If possible, store your files to disks that are not used by other processes, including virtual memory or swap space. If you have multiple disks available, for the best performance, place one directory on each disk.
    - Use different directories for different members. You can use any number of directories for a single disk store.

Create and Configure Your Disk Stores

1. In the locations you have chosen, create all directories you will specify for your disk stores to use. VMware GemFire throws an exception if the specified directories are not available when a disk store is created. You do not need to populate these directories with anything.
2. Open a `gfsh` prompt and connect to the cluster.
3. At the `gfsh` prompt, create and configure a disk store:
    - Specify the name (`--name`) of the disk-store.
        - Choose disk store names that reflect how the stores should be used and that work for your operating systems. Disk store names are used in the disk file names:
            - Use disk store names that satisfy the file naming requirements for your operating system. For example, if you store your data to disk in a Windows system, your disk store names could not contain any of these reserved characters, < > : " / \ | ? *.
            - Do not use very long disk store names. The full file names must fit within your operating system limits. On Linux, for example, the standard limitation is 255 characters.

    ```
    gfsh>create disk-store --name=serverOverflow --dir=c:\overflow_data#20480
    ```

    - Configure the directory locations (`--dir`) and the maximum space to use for the store (specified after the disk directory name by # and the maximum number in megabytes).

    ```
    gfsh>create disk-store --name=serverOverflow --dir=c:\overflow_data#20480
    ```

    - Optionally, you can configure the store's file compaction behavior. In conjunction with this, plan and program for any manual compaction. Example:

    ```
    gfsh>create disk-store --name=serverOverflow --dir=c:\overflow_data#20480 
    \
    ```

```
--compaction-threshold=40 --auto-compact=false --allow-force-compaction=t
rue
```

- If needed, configure the maximum size (in MB) of a single oplog. When the current files reach this size, the system rolls forward to a new file. You get better performance with relatively small maximum file sizes. Example:

  ```
  gfsh>create disk-store --name=serverOverflow --dir=c:\overflow_data#20480
  \
  --compaction-threshold=40 --auto-compact=false --allow-force-compaction=t
  rue \
  --max-oplog-size=512
  ```

- If needed, modify queue management parameters for asynchronous queueing to the disk store. You can configure any region for synchronous or asynchronous queueing (region attribute `disk-synchronous`). Server queues and gateway sender queues always operate synchronously. When either the `queue-size` (number of operations) or `time-interval` (milliseconds) is reached, enqueued data is flushed to disk. You can also synchronously flush unwritten data to disk through the `DiskStore flushToDisk` method. Example:

  ```
  gfsh>create disk-store --name=serverOverflow --dir=c:\overflow_data#20480
  \
  --compaction-threshold=40 --auto-compact=false --allow-force-compaction=t
  rue \
  --max-oplog-size=512 --queue-size=10000 --time-interval=15
  ```

- If needed, modify the size (specified in bytes) of the buffer used for writing to disk. Example:

  ```
  gfsh>create disk-store --name=serverOverflow --dir=c:\overflow_data#20480
  \
  --compaction-threshold=40 --auto-compact=false --allow-force-compaction=t
  rue \
  --max-oplog-size=512 --queue-size=10000 --time-interval=15 --write-buffer
  -size=65536
  ```

- If needed, modify the `disk-usage-warning-percentage` and `disk-usage-critical-percentage` thresholds that determine the percentage (default: 90%) of disk usage that will trigger a warning and the percentage (default: 99%) of disk usage that will generate an error and shut down the member cache. Example:

  ```
  gfsh>create disk-store --name=serverOverflow --dir=c:\overflow_data#20480
  \
  --compaction-threshold=40 --auto-compact=false --allow-force-compaction=t
  rue \
  --max-oplog-size=512 --queue-size=10000 --time-interval=15 --write-buffer
  -size=65536 \
  --disk-usage-warning-percentage=80 --disk-usage-critical-percentage=98
  ```

The following is the complete disk store cache.xml configuration example:

```
<disk-store name="serverOverflow" compaction-threshold="40"
           auto-compact="false" allow-force-compaction="true"
```

```
         max-oplog-size="512" queue-size="10000"
         time-interval="15" write-buffer-size="65536"
         disk-usage-warning-percentage="80"
         disk-usage-critical-percentage="98">
        <disk-dirs>
               <disk-dir>c:\overflow_data</disk-dir>
               <disk-dir dir-size="20480">d:\overflow_data</disk-dir>
        </disk-dirs>
</disk-store>
```

Note: As an alternative to defining cache.xml on every server in the cluster– if you have the cluster configuration service enabled, when you create a disk store in `gfsh`, you can share the disk store's configuration with the rest of cluster. See Overview of the Cluster Configuration Service.

Modifying Disk Stores

You can modify an offline disk store by using the alter disk-store command. If you are modifying the default disk store configuration, use “DEFAULT” as the disk-store name.

Configuring Regions, Queues, and PDX Serialization to Use the Disk Stores

The following are examples of using already created and named disk stores for Regions, Queues, and PDX Serialization.

Example of using a disk store for region persistence and overflow:

- gfsh:

  ```
  gfsh>create region --name=regionName --type=PARTITION_PERSISTENT_OVERFLOW \
  --disk-store=serverPersistOverflow
  ```

- cache.xml

  ```
  <region refid="PARTITION_PERSISTENT_OVERFLOW" disk-store-name="persistOverflow
  1"/>
  ```

Example of using a named disk store for server subscription queue overflow (cache.xml):

```
<cache-server port="40404">
   <client-subscription
      eviction-policy="entry"
      capacity="10000"
      disk-store-name="queueOverflow2"/>
</cache-server>
```

Example of using a named disk store for PDX serialization metadata (cache.xml):

```
<pdx read-serialized="true"
    persistent="true"
    disk-store-name="SerializationDiskStore">
</pdx>
```

Configuring Disk Stores on Gateway Senders

Gateway sender queues are always overflowed and may be persisted. Assign them to overflow disk stores if you do not persist, and to persistence disk stores if you do.

Persisted data from a parallel gateway sender must go to the same disk store as used by the region, because parallel gateway sender queues must be colocated with their regions to operate correctly.

Example of using a named disk store for a serial gateway sender queue persistence:

- gfsh:

  ```
  gfsh>create gateway-sender --id=persistedSender1 --remote-distributed-system-id
  =1 \
  --enable-persistence=true --disk-store-name=diskStoreA --maximum-queue-memory=1
  00
  ```

- cache.xml:

  ```
  <cache>
    <gateway-sender id="persistedsender1" parallel="true"
     remote-distributed-system-id="1"
     enable-persistence="true"
     disk-store-name="diskStoreA"
     maximum-queue-memory="100"/>
     ...
  </cache>
  ```

Examples of using the default disk store for a serial gateway sender queue persistence and overflow:

- gfsh:

  ```
  gfsh>create gateway-sender --id=persistedSender1 --remote-distributed-system-id
  =1 \
  --enable-persistence=true --maximum-queue-memory=100
  ```

- cache.xml:

  ```
  <cache>
    <gateway-sender id="persistedsender1" parallel="true"
     remote-distributed-system-id="1"
     enable-persistence="true"
     maximum-queue-memory="100"/>
     ...
  </cache>
  ```

Disk Store Configuration Parameters

You define your disk stores by using the `gfsh create disk-store` command or in `<disk-store>` subelements of your cache declaration in `cache.xml`. All disk stores are available for use by all of your regions and queues.

These `<disk-store>` attributes and subelements have corresponding `gfsh create disk-store` command-line parameters as well as getter and setter methods in the `org.apache.geode.cache.DiskStoreFactory` and `org.apache.geode.cache.DiskStore` APIs.

Disk Store Configuration Attributes and Elements

disk-store attribute	Description	Default
`name`	String used to identify this disk store. All regions and queues select their disk store by specifying this name.	DEFAULT
`allow-force-compaction`	Boolean indicating whether to allow manual compaction through the API or command-line tools.	false
`auto-compact`	Boolean indicating whether to automatically compact a file when its live data content percentage drops below the `compaction-threshold`.	true
`compaction-threshold`	Percentage (0..100) of live data (non-garbage content) remaining in the operation log, below which it is eligible for compaction. As garbage is created (by entry destroys, entry updates, and region destroys and creates) the percentage of remaining live data declines. Falling below this percentage initiates compaction if auto-compaction is turned on. If not, the file will be eligible for manual compaction at a later time.	50
`disk-usage-critical-percentage`	Disk usage above this threshold generates an error message and shuts down the member's cache. For example, if the threshold is set to 99%, then falling under 10 GB of free disk space on a 1 TB drive generates the error and shuts down the cache. Set to "0" (zero) to disable.	99
`disk-usage-warning-percentage`	Disk usage above this threshold generates a warning message. For example, if the threshold is set to 90%, then on a 1 TB drive falling under 100 GB of free disk space generates the warning. Set to "0" (zero) to disable.	90
`max-oplog-size`	The largest size, in megabytes, to allow an operation log to become before automatically rolling to a new file. This size is the combined sizes of the oplog files.	1024

disk-store attribute	Description	Default
`queue-size`	For asynchronous queueing. The maximum number of operations to allow into the write queue before automatically flushing the queue. Operations that would add entries to the queue block until the queue is flushed. A value of zero implies no size limit. Reaching this limit or the time-interval limit will cause the queue to flush.	0
`time-interval`	For asynchronous queueing. The number of milliseconds that can elapse before data is flushed to disk. Reaching this limit or the queue-size limit causes the queue to flush.	1000
`write-buffer-size`	Size of the buffer, in bytes, used to write to disk.	32768

`disk-store` **subelement**	Description	Default
`<disk-dirs>`	Defines the system directories where the disk store is written and their maximum sizes.	`.` with no size limit

disk-dirs Element

The `<disk-dirs>` element defines the host system directories to use for the disk store. It contains one or more single `<disk-dir>` elements with the following contents:

- The directory specification, provided as the text of the `disk-dir` element.
- An optional `dir-size` attribute specifying the maximum amount of space, in megabytes, to use for the disk store in the directory. By default, there is no limit. The space used is calculated as the combined sizes of all oplog files.

You can specify any number of `disk-dir` subelements to the `disk-dirs` element. The data is spread evenly among the active disk files in the directories, keeping within any limits you set.

Example:

```
<disk-dirs>
    <disk-dir>/host1/users/gf/memberA_DStore</disk-dir>
    <disk-dir>/host2/users/gf/memberA_DStore</disk-dir>
    <disk-dir dir-size="20480">/host3/users/gf/memberA_DStore</disk-dir>
</disk-dirs>
```

Note: The directories must exist when the disk store is created or the system throws an exception. VMware GemFire does not create directories.

Use different disk-dir specifications for different disk stores. You cannot use the same directory for the same named disk store in two different members.

Modifying the Default Disk Store

You can modify the behavior of the default disk store by specifying the attributes you want for the disk store named “DEFAULT”.

Whenever you use disk stores without specifying the disk store to use, VMware GemFire uses the disk store named “DEFAULT”.

For example, these region and queue configurations specify persistence and/or overflow, but do not specify the disk-store-name. Because no disk store is specified, these use the disk store named “DEFAULT”.

Examples of using the default disk store for region persistence and overflow:

- gfsh:

  ```
  gfsh>create region --name=regionName --type=PARTITION_PERSISTENT_OVERFLOW
  ```

- cache.xml

  ```
  <region refid="PARTITION_PERSISTENT_OVERFLOW"/>
  ```

Example of using the default disk store for server subscription queue overflow (cache.xml):

```
<cache-server port="40404">
    <client-subscription eviction-policy="entry" capacity="10000"/>
</cache-server>
```

Change the Behavior of the Default Disk Store

VMware GemFire initializes the default disk store with the default disk store configuration settings. You can modify the behavior of the default disk store by specifying the attributes you want for the disk store named “DEFAULT”. The only thing you can’t change about the default disk store is the name.

The following example changes the default disk store to allow manual compaction and to use multiple, non-default directories:

cache.xml:

```
<disk-store name="DEFAULT" allow-force-compaction="true">
     <disk-dirs>
        <disk-dir>/export/thor/customerData</disk-dir>
        <disk-dir>/export/odin/customerData</disk-dir>
        <disk-dir>/export/embla/customerData</disk-dir>
     </disk-dirs>
</disk-store>
```

Optimizing a System with Disk Stores

Optimize availability and performance by following the guidelines in this section.

1. VMware GemFire recommends the use of `ext4` filesystems when operating on Linux or Solaris platforms. The `ext4` filesystem supports preallocation, which benefits disk startup performance. If you are using `ext3` filesystems in latency-sensitive environments with high write throughput, you can improve disk startup performance by setting the `maxOplogSize`

(see the `DiskStoreFactory.setMaxOplogSize`) to a value lower than the default 1 GB and by disabling preallocation by specifying the system property `gemfire.preAllocateDisk=false` upon VMware GemFire process startup.

2. When you start your system, start all the members that have persistent regions at roughly the same time. Create and use startup scripts for consistency and completeness.
3. Shut down your system using the gfsh `shutdown` command. This is an ordered shutdown that positions your disk stores for a faster startup.
4. Configure critical usage thresholds (`disk-usage-warning-percentage` and `disk-usage-critical-percentage`) for the disk. By default, these are set to 80% for warning and 99% for errors that will shut down the cache.
5. Decide on a file compaction policy and, if needed, develop procedures to monitor your files and execute regular compaction.
6. Decide on a backup strategy for your disk stores and follow it. You can back up a running sytem by using the `backup disk-store` command.
7. If you remove any persistent region or change its configuration while your disk store is offline, consider synchronizing the regions in your disk stores.

Start Up and Shut Down with Disk Stores

This section describes what happens during startup and shutdown and provides procedures for those operations.

Start Up

When you start a member with a persistent region, the data is retrieved from disk stores to recreate the member's persistent region. If the member does not hold all of the most recent data for the region, then other members have the data, and region creation blocks, waiting for the those other members. A partitioned region with colocated entries also blocks on start up, waiting for the entries of the colocated region to be available. A persistent gateway sender is treated the same as a colocated region, so it can also block region creation.

With a log level of info or below, the system provides messaging about the wait. Here, the disk store for server2 has the most recent data for the region, and server1 is waiting for server2.

```
Region /people has potentially stale data.
It is waiting for another member to recover the latest data.
My persistent id:

  DiskStore ID: 6893751ee74d4fbd-b4780d844e6d5ce7
  Name: server1
  Location: /192.0.2.0:/home/dsmith/server1/.

Members with potentially new data:
[
  DiskStore ID: 160d415538c44ab0-9f7d97bae0a2f8de
  Name: server2
  Location: /192.0.2.0:/home/dsmith/server2/.
]
```

```
Use the "gfsh show missing-disk-stores" command to see all disk stores
that are being waited on by other members.
```

When the most recent data is available, the system updates the region, logs a message, and continues the startup.

```
[info 2010/04/09 10:52:13.010 PDT CacheRunner <main> tid=0x1]
   Done waiting for the remote data to be available.
```

If the member's disk store has data for a region that is never created, the data remains in the disk store.

Each member's persistent regions load and go online as quickly as possible, not waiting unnecessarily for other members to complete. For performance reasons, these actions occur asynchronously:

- Once at least one copy of each and every bucket is recovered from disk, the region is available. Secondary buckets will load asynchronously.
- Entry keys are loaded from the key file in the disk store before considering entry values. Once all keys are loaded, VMware GemFire loads the entry values asynchronously. If a value is requested before it has loaded, the value will immediately be fetched from the disk store.

Start Up Procedure

To start a system with disk stores:

1. **Start all members with persisted data first and at the same time**. Exactly how you do this depends on your members. Make sure to start members that host colocated regions, as well as persistent gateway senders.

   While they are initializing their regions, the members determine which have the most recent region data, and initialize their regions with the most recent data.

   For replicated regions, where you define persistence only in some of the region's host members, start the persistent replicate members prior to the non-persistent replicate members to make sure the data is recovered from disk.

   This is an example bash script for starting members in parallel. The script waits for the startup to finish. It exits with an error status if one of the jobs fails.

   ```
   #!/bin/bash
   ssh servera "cd /my/directory; gfsh start server --name=servera &
   ssh serverb "cd /my/directory; gfsh start server --name=serverb &

   STATUS=0;
   for job in `jobs -p`
   do
   echo $job
   wait $job;
   JOB_STATUS=$?;
   test $STATUS -eq 0 && STATUS=$JOB_STATUS;
   done
   exit $STATUS;
   ```

2. **Respond to blocked members**. When a member blocks waiting for more recent data from another member, the member waits indefinitely rather than coming online with stale data. Check for missing disk stores with the `gfsh show missing-disk-stores` command. See Handling Missing Disk Stores.
    - If no disk stores are missing, the cache initialization must be slow for some other reason. Check the information on member hangs in Diagnosing System Problems.
    - If disk stores are missing that you think should be there:
        - Make sure you have started the member. Check the logs for any failure messages. See Logging.
        - Make sure your disk store files are accessible. If you have moved your member or disk store files, you must update your disk store configuration to match.
    - If disk stores are missing that you know are lost, because you have deleted them or their files are otherwise unavailable, revoke them so the startup can continue.

Example Startup to Illustrate Ordering

The following lists the two possibilities for starting up a replicated persistent region after a shutdown. Assume that Member A (MA) exits first, leaving persisted data on disk for RegionP. Member B (MB) continues to run operations on RegionP, which update its disk store and leave the disk store for MA in a stale condition. MB exits, leaving the most up-to-date data on disk for RegionP.

- Restart order 1
    1. MB is started first. MB identifies that it has the most recent disk data for RegionP and initializes the region from disk. MB does not block.
    2. MA is started, recovers its data from disk, and updates region data as needed from the data in MB.
- Restart order 2
    1. MA is started first. MA identifies that it does not have the most recent disk data and blocks, waiting for MB to start before recreating RegionP in MA.
    2. MB is started. MB identifies that it has the most recent disk data for RegionP and initializes the region from disk.
    3. MA recovers its RegionP data from disk and updates region data as needed from the data in MB.

Shutdown

If more than one member hosts a persistent region or queue, the order in which the various members shut down may be significant upon restart of the system. The last member to exit the system or shut down has the most up-to-date data on disk. Each member knows which other system members were online at the time of exit or shutdown. This permits a member to acquire the most recent data upon subsequent start up.

For a replicated region with persistence, the last member to exit has the most recent data.

For a partitioned region every member persists its own buckets. A shutdown using `gfsh shutdown` will synchronize the disk stores before exiting, so all disk stores hold the most recent data. Without an orderly shutdown, some disk stores may have more recent bucket data than others.

The best way to shut down a system is to invoke the `gfsh shutdown` command with all members running. All online data stores will be synchronized before shutting down, so all hold the most recent data copy. To shut down all members other than locators:

```
gfsh>shutdown
```

To shut down all members, including locators:

```
gfsh>shutdown --include-locators=true
```

Disk Store Management

The `gfsh` command-line tool has a number of options for examining and managing your disk stores. The `gfsh` tool, the `cache.xml` file and the DiskStore APIs are your management tools for online and offline disk stores.

See Disk Store Commands for a list of available commands.

- **Disk Store Management Commands and Operations**
- **Validating a Disk Store**
- **Running Compaction on Disk Store Log Files**
- **Keeping a Disk Store Synchronized with the Cache**
- **Configuring Disk Free Space Monitoring**
- **Handling Missing Disk Stores**
- **Altering When Buffers Are Flushed to Disk**

  You can configure VMware GemFire to write immediately to disk and you may be able to modify your operating system behavior to perform buffer flushes more frequently.

Disk Store Management Commands and Operations

You can manage your disk stores using the gfsh command-line tool. For more information on `gfsh` commands, see gfsh and Disk Store Commands.

Note: Each of these commands operates either on the online disk stores or offline disk stores, but not both.

gfsh Command	Online or Offline Command	See ...
`alter disk-store`	Off	Keeping a Disk Store Synchronized with the Cache
`compact disk-store`	On	Running Compaction on Disk Store Log Files
`backup disk-store`	On	Creating Backups for System Recovery and Operational Management

gfsh Command	Online or Offline Command	See …
`compact offline-disk-store`	Off	Running Compaction on Disk Store Log Files
`export offline-disk-store`	Off	Creating Backups for System Recovery and Operational Management
`revoke missing-disk-store`	On	Handling Missing Disk Stores
`show missing-disk-stores`	On	Handling Missing Disk Stores
`shutdown`	On	Start Up and Shut Down with Disk Stores
`validate offline disk-store`	Off	Validating a Disk Store

For complete command syntax of any gfsh command, run `help <command>` at the gfsh command line.

Online Disk Store Operations

For online operations, `gfsh` must be connected to a cluster via a JMX manager and sends the operation requests to the members that have disk stores. These commands will not run on offline disk stores.

Offline Disk Store Operations

For offline operations, `gfsh` runs the command against the specified disk store and its specified directories. You must specify all directories for the disk store. For example:

```
gfsh>compact offline-disk-store --name=mydiskstore --disk-dirs=MyDirs
```

Offline operations will not run on online disk stores. The tool locks the disk store while it is running, so the member cannot start in the middle of an operation.

If you try to run an offline command for an online disk store, you get a message like this:

```
gfsh>compact offline-disk-store --name=DEFAULT --disk-dirs=s1
This disk store is in use by another process. "compact disk-store" can
be used to compact a disk store that is currently in use.
```

Validating a Disk Store

The `validate offline-disk-store` command verifies the health of your offline disk store and gives you information about the regions in it, the total entries, and the number of records that would be removed if you compacted the store.

Use this command at these times:

- Before compacting an offline disk store to help decide whether it's worth doing.
- Before restoring or modifying a disk store.

- Any time you want to be sure the disk store is in good shape.

Example:

```
gfsh>validate offline-disk-store --name=ds1 --disk-dirs=hostB/bupDirectory
```

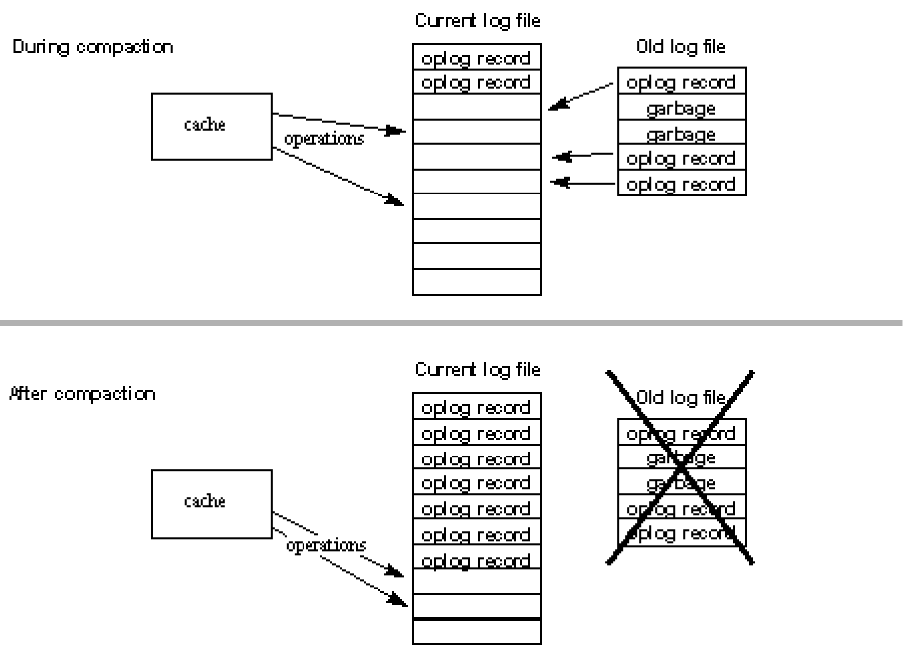
Running Compaction on Disk Store Log Files

When a cache operation is added to a disk store, any preexisting operation record for the same entry becomes obsolete, and VMware GemFire marks it as garbage. For example, when you create an entry, the create operation is added to the store. If you update the entry later, the update operation is added and the create operation becomes garbage. VMware GemFire does not remove garbage records as it goes, but it tracks the percentage of non-garbage (live data) remaining in each operation log, and provides mechanisms for removing garbage to compact your log files.

VMware GemFire compacts an old operation log by copying all non-garbage records into the current log and discarding the old files. As with logging, oplogs are rolled as needed during compaction to stay within the max oplog setting.

The system is configured by default to automatically compact any closed operation log when its non-garbage content drops below a certain percentage. This automatic compaction is well suited to most VMware GemFire implementations. In some circumstances, you may choose to manually initiate compaction for online and offline disk stores.

Log File Compaction for the Online Disk Store



For the online disk store, the current operation log is not available for compaction, no matter how much garbage it contains. You can use `DiskStore.forceRoll` to close the current oplog, making it eligible for compaction. See Disk Store Operation Logs for details.

Offline compaction runs essentially in the same way, but without the incoming cache operations. Also, because there is no currently open log, the compaction creates a new one to get started.

Run Online Compaction

Old log files become eligible for online compaction when their live data (non-garbage) content drops below a configured percentage of the total file. A record is garbage when its operation is superseded by a more recent operation for the same object. During compaction, the non-garbage records are added to the current log along with new cache operations. Online compaction does not block current system operations.

- **Automatic compaction**. When `auto-compact` is true, VMware GemFire automatically compacts each oplog when its non-garbage (live data) content drops below the `compaction-threshold`. This takes cycles from your other operations, so you may want to deactivate this and only do manual compaction, to control the timing.
- **Manual compaction**. To run manual compaction:
    - Set the disk store attribute `allow-force-compaction` to true. This causes VMware GemFire to maintain extra data about the files so it can compact on demand. This is deactivated by default to save space. You can run manual online compaction at any time while the system is running. Oplogs eligible for compaction based on the `compaction-threshold` are compacted into the current oplog.
    - Run manual compaction as needed. VMware GemFire has two types of manual compaction:
        - Compact the logs for a single online disk store through the API, with the `forceCompaction` method. This method first rolls the oplogs and then compacts them. Example:

          ```
          myCache.findDiskStore("myDiskStore").forceCompaction();
          ```

        - Using `gfsh`, compact a disk store with the compact disk-store command. Examples:

          ```
          gfsh>compact disk-store --name=Disk1

          gfsh>compact disk-store --name=Disk1 --group=MemberGroup1,MemberGr
          oup2
          ```

          Note: You need to be connected to a JMX Manager in `gfsh` to run this command.

Run Offline Compaction

Offline compaction is a manual process. All log files are compacted as much as possible, regardless of how much garbage they hold. Offline compaction creates new log files for the compacted log records.

Using `gfsh`, compact individual offline disk stores with the compact offline-disk-store command:

```
gfsh>compact offline-disk-store --name=Disk2 --disk-dirs=/Disks/Disk2

gfsh>compact offline-disk-store --name=Disk2 --disk-dirs=/Disks/Disk2
--max-oplog-size=512 -J=-Xmx1024m
```

Note: Do not perform offline compaction on the baseline directory of an incremental backup.

You must provide all of the directories in the disk store. If no oplog max size is specified, VMware GemFire uses the system default.

Offline compaction can take a lot of memory. If you get a `java.lang.OutOfMemory` error while running this, you may need to increase your heap size with the `-J=-Xmx` parameter.

Performance Benefits of Manual Compaction

You can improve performance during busy times if you deactivate automatic compaction and run your own manual compaction during lighter system load or during downtimes. You could run the API call after your application performs a large set of data operations. You could run `compact disk-store` command every night when system use is very low.

To follow a strategy like this, you need to set aside enough disk space to accommodate all non-compacted disk data. You might need to increase system monitoring to make sure you do not overrun your disk space. You may be able to run only offline compaction. If so, you can set `allow-force-compaction` to false and avoid storing the information required for manual online compaction.

Directory Size Limits

Reaching directory size limits during compaction has different results depending on whether you are running an automatic or manual compaction:

- For automatic compaction, the system logs a warning, but does not stop.
- For manual compaction, the operation stops and returns a `DiskAccessException` to the calling process, reporting that the system has run out of disk space.

Example Compaction Run

In this example offline compaction run listing, the disk store compaction had nothing to do in the `*_3.*` files, so they were left alone. The `*_4.*` files had garbage records, so the oplog from them was compacted into the new `*_5.*` files.

```
bash-2.05$ ls -ltra backupDirectory
total 28
-rw-rw-r--   1 user users          3 Apr  7 14:56 BACKUPds1_3.drf
-rw-rw-r--   1 user users         25 Apr  7 14:56 BACKUPds1_3.crf
drwxrwxr-x   3 user users       1024 Apr  7 15:02 ..
-rw-rw-r--   1 user users       7085 Apr  7 15:06 BACKUPds1.if
-rw-rw-r--   1 user users         18 Apr  7 15:07 BACKUPds1_4.drf
-rw-rw-r--   1 user users       1070 Apr  7 15:07 BACKUPds1_4.crf
drwxrwxr-x   2 user users        512 Apr  7 15:07 .

bash-2.05$ gfsh

gfsh>validate offline-disk-store --name=ds1 --disk-dirs=backupDirectory

/root: entryCount=6
/partitioned_region entryCount=1 bucketCount=10
Disk store contains 12 compactable records.
Total number of region entries in this disk store is: 7
```

```
gfsh>compact offline-disk-store --name=ds1 --disk-dirs=backupDirectory
Offline compaction removed 12 records.
Total number of region entries in this disk store is: 7

gfsh>exit

bash-2.05$ ls -ltra backupDirectory
total 16
-rw-rw-r--   1 user users          3 Apr  7 14:56 BACKUPds1_3.drf
-rw-rw-r--   1 user users         25 Apr  7 14:56 BACKUPds1_3.crf
drwxrwxr-x   3 user users       1024 Apr  7 15:02 ..
-rw-rw-r--   1 user users          0 Apr  7 15:08 BACKUPds1_5.drf
-rw-rw-r--   1 user users        638 Apr  7 15:08 BACKUPds1_5.crf
-rw-rw-r--   1 user users       2788 Apr  7 15:08 BACKUPds1.if
drwxrwxr-x   2 user users        512 Apr  7 15:09 .
bash-2.05$
```

Keeping a Disk Store Synchronized with the Cache

Recovering data from an offline disk store proceeds most quickly when the configuration of the offline data matches that of the online data.

Whenever you change or remove persistent regions (by modifying your cache.xml or the code that configures the regions), then you should alter the corresponding offline disk-store to match. If you don’t, then the next time this disk-store is recovered it will recover all of that region’s data into a temporary region using the old configuration. The old configuration will still consume the old configured resources (heap memory, off-heap memory). If those resources are no longer available (for example the old configuration of the region was off-heap but you decide to no longer configure off-heap memory on the JVM), the disk-store recovery will fail.

It is common practice to have more than one off-line disk store, because each member of the cluster usually has its own copy. Be sure to apply the same `alter disk-store` command to each offline copy of the disk store.

Change Region Configuration

When your disk store is offline, you can keep the configuration for its regions up-to-date with your `cache.xml` and API settings. The disk store retains a subset of the region configuration attributes. (For a list of the retained attributes, see alter disk-store). If the configurations do not match at startup, the `cache.xml` and API override any disk store settings and the disk store is automatically updated to match. So you do not need to modify your disk store to keep your cache configuration and disk store synchronized, but you will save startup time and memory if you do.

For example, to change the initial capacity of the region named “partitioned_region” in the disk store:

```
gfsh>alter disk-store --name=myDiskStoreName --region=partitioned_region
--disk-dirs=/firstDiskStoreDir,/secondDiskStoreDir,/thirdDiskStoreDir
--initialCapacity=20
```

To list all modifiable settings and their current values for a region, run the command with no actions specified:

```
gfsh>alter disk-store --name=myDiskStoreName --region=partitioned_region
--disk-dirs=/firstDiskStoreDir,/secondDiskStoreDir,/thirdDiskStoreDir
```

Take a Region Out of Your Cache Configuration and Disk Store

You might remove a region from your application if you decide to rename it or to split its data into two entirely different regions. Any significant data restructuring can cause you to retire some data regions.

This applies to the removal of regions while the disk store is offline. Regions you destroy through API calls or by `gfsh` are automatically removed from the disk store of online members.

In your application development, when you discontinue use of a persistent region, remove the region from the member’s disk store as well.

Note: Perform the following operations with caution. You are permanently removing data.

You can remove the region from the disk store in one of two ways:

- Delete the entire set of disk store files. Your member will initialize with an empty set of files the next time you start it. Exercise caution when removing the files from the file system, as more than one region can be specified to use the same disk store directories.
- Selectively remove the discontinued region from the disk store with a command such as:

  ```
  gfsh>alter disk-store --name=myDiskStoreName --region=partitioned_region
  --disk-dirs=/firstDiskStoreDir,/secondDiskStoreDir,/thirdDiskStoreDir --remove
  ```

To guard against unintended data loss, VMware GemFire maintains the region in the disk store until you manually remove it. Regions in the disk stores that are not associated with any region in your application are still loaded into temporary regions in memory and kept there for the life of the member. The system has no way of detecting whether the cache region will be created by your API at some point, so it keeps the temporary region loaded and available.

Configuring Disk Free Space Monitoring

To modify `disk-usage-warning-percentage` and `disk-usage-critical-percentage` thresholds, specify the parameters when executing the `gfsh create disk-store` command.

```
gfsh>create disk-store --name=serverOverflow --dir=c:\overflow_data#20480 \
--compaction-threshold=40 --auto-compact=false --allow-force-compaction=true \
--max-oplog-size=512 --queue-size=10000 --time-interval=15 --write-buffer-size=65536 \
--disk-usage-warning-percentage=80 --disk-usage-critical-percentage=98
```

By default, disk usage above 80% triggers a warning message. Disk usage above 99% generates an error and shuts down the member cache that accesses that disk store. To disable disk store monitoring, set the parameters to 0.

To view the current threshold values set for an existing disk store, use the `gfsh describe` disk-store command:

```
gfsh>describe disk-store --member=server1 --name=DiskStore1
```

You can also use the following `DiskStoreMXBean` method APIs to configure and obtain these thresholds programmatically.

- `getDiskUsageCriticalPercentage`
- `getDiskUsageWarningPercentage`
- `setDiskUsageCriticalPercentage`
- `setDiskUsageWarningPercentage`

You can obtain statistics on disk space usage and the performance of disk space monitoring by accessing the following statistics:

- `diskSpace`
- `maximumSpace`
- `volumeSize`
- `volumeFreeSpace`
- `volumeFreeSpaceChecks`
- `volumeFreeSpaceTime`

See Disk Space Usage (DiskDirStatistics).

Handling Missing Disk Stores

This section applies to disk stores that hold the latest copy of your data for at least one region.

Show Missing Disk Stores

Using `gfsh`, the `show missing-disk-stores` command lists all disk stores with most recent data that are being waited on by other members.

For replicated regions, this command only lists missing members that are preventing other members from starting up. For partitioned regions, this command also lists any offline data stores, even when other data stores for the region are online, because their offline status may be causing `PartitionOfflineExceptions` in cache operations or preventing the system from satisfying redundancy.

Example:

```
gfsh>show missing-disk-stores
          Disk Store ID              |   Host    |               Directory
------------------------------------ | --------- | ------------------------------------
--
60399215-532b-406f-b81f-9b5bd8d1b55a | excalibur | /usr/local/gemfire/deploy/disk_stor
e1
```

Note: You need to be connected to JMX Manager in `gfsh` to run this command.

Note: The disk store directories listed for missing disk stores may not be the directories you have currently configured for the member. The list is retrieved from the other running members—the ones who are reporting the missing member. They have information from the last time the missing disk store was online. If you move your files and change the member's configuration, these directory locations will be stale.

Disk stores usually go missing because their member fails to start. The member can fail to start for a number of reasons, including:

- Disk store file corruption. You can check on this by validating the disk store.
- Incorrect cluster configuration for the member
- Network partitioning
- Drive failure

Revoke Missing Disk Stores

This section applies to disk stores for which both of the following are true:

- Disk stores that have the most recent copy of data for one or more regions or region buckets.
- Disk stores that are unrecoverable, such as when you have deleted them, or their files are corrupted or on a disk that has had a catastrophic failure.

When you cannot bring the latest persisted copy online, use the revoke command to tell the other members to stop waiting for it. Once the store is revoked, the system finds the remaining most recent copy of data and uses that.

Note: Once revoked, a disk store cannot be reintroduced into the system.

Use gfsh show missing-disk-stores to properly identify the disk store you need to revoke. The revoke command takes the disk store ID as input, as listed by that command.

Example:

```
gfsh>revoke missing-disk-store --id=60399215-532b-406f-b81f-9b5bd8d1b55a
Missing disk store successfully revoked
```

Altering When Buffers Are Flushed to Disk

You can configure VMware GemFire to write immediately to disk and you may be able to modify your operating system behavior to perform buffer flushes more frequently.

Typically, VMware GemFire writes disk data into the operating system's disk buffers and the operating system periodically flushes the buffers to disk. Increasing the frequency of writes to disk decreases the likelihood of data loss from application or machine crashes, but it impacts performance. Your other option, which may give you better performance, is to use VMware GemFire's in-memory data backups. Do this by storing your data in multiple replicated regions or in partitioned regions that are configured with redundant copies. See Region Types.

Modifying Disk Flushes for the Operating System

You may be able to change the operating system settings for periodic flushes. You may also be able to perform explicit disk flushes from your application code. For information on these options, see your operating system's documentation. For example, in Linux you can change the disk flush interval by modifying the setting `/proc/sys/vm/dirty_expire_centiseconds`. It defaults to 30 seconds. To alter this setting, see the Linux documentation for `dirty_expire_centiseconds`.

Modifying VMware GemFire to Flush Buffers on Disk Writes

You can have VMware GemFire flush the disk buffers on every disk write. Do this by setting the system property `gemfire.syncWrites` to true at the command line when you start your VMware GemFire member. You can only modify this setting when you start a member. When this is set, VMware GemFire uses a Java `RandomAccessFile` with the flags "rwd", which causes every file update to be written synchronously to the storage device. This only guarantees your data if your disk stores are on a local device. See the Java documentation for `java.IO.RandomAccessFile`.

To modify the setting for a VMware GemFire application, add this to the java command line when you start the member:

```
-Dgemfire.syncWrites=true
```

To modify the setting for a cache server, use this syntax:

```
gfsh>start server --name=... --J=-Dgemfire.syncWrites=true
```

Creating Backups for System Recovery and Operational Management

A backup is a copy of persisted data from a disk store. A backup is used to restore the disk store to the state it was in when the backup was made. The appropriate back up and restore procedures differ based upon whether the cluster is online or offline. An online system has currently running members. An offline system does not have any running members.

- Making a Backup While the System Is Online
- What a Full Online Backup Saves
- What an Incremental Online Backup Saves
- Disk Store Backup Directory Structure and Contents
- Offline Members—Manual Catch-Up to an Online Backup
- Restore Using a Backup Made While the System Was Online

Making a Backup While the System Is Online

The gfsh command `backup disk-store` creates a backup of the disk stores for all members running in the cluster. The backup works by passing commands to the running system members; therefore, the members need to be online for this operation to succeed. Each member with persistent data creates a backup of its own configuration and disk stores. The backup does not block any activities within the cluster, but it does use resources.

Note: Do not try to create backup files from a running system by using your operating system's file copy commands. This would create incomplete and unusable copies.

Preparing to Make a Backup

- Consider compacting your disk store before making a backup. If auto-compaction is turned off, you may want to do a manual compaction to save on the quantity of data copied over the network by the backup. For more information on configuring a manual compaction, see Manual Compaction.
- Take the backup when region operations are quiescent, to avoid the possibility of an inconsistency between region data and an asynchronous event queue (AEQ) or a WAN Gateway sender (which uses a persistent queue). A region operation that causes a persisted write to a region involves a disk operation. The associated queue operation also causes a disk operation. These two disk operations are not made atomically, so if a backup is made between the two disk operations, then the backup represents inconsistent data in the region and the queue.
- Run the backup during a period of low activity in your system. The backup does not block system activities, but it uses file system resources on all hosts in your cluster, and it can affect performance.
- Configure each member with any additional files or directories to be backed up by modifying the member's `cache.xml` file. Additional items that ought to be included in the backup:
    - application jar files
    - other files that the application needs when starting, such as a file that sets the classpath

  For example, to include file `myExtraBackupStuff` in the backup, the `cache.xml` file specification of the data store would include:

  ```
  <backup>./myExtraBackupStuff</backup>
  ```

  Directories are recursively copied, with any disk stores that are found excluded from this user-specified backup.
- Back up to a SAN (recommended) or to a directory that all members can access. Make sure the directory exists and has the proper permissions for all members to write to the directory and create subdirectories.

  The directory specified for the backup can be used multiple times. Each time a backup is made, a new subdirectory is created within the specified directory, and that new subdirectory's name represents the date and time.

  You can use one of two locations for the backup:
    - a single physical location, such as a network file server, for example:

      ```
      /export/fileServerDirectory/gemfireBackupLocation
      ```
    - a directory that is local to all host machines in the system, for example:

      ```
      ./gemfireBackupLocation
      ```

- Make sure all members with persistent data are running in the system, because offline members cannot back up their disk stores. Output from the backup command will not identify members hosting replicated regions that are offline.

How to Do a Full Online Backup

1. If auto-compaction is disabled, and manual compaction is needed:

   ```
   gfsh>compact disk-store --name=Disk1
   ```

2. Run the `gfsh backup disk-store` command, specifying the backup directory location. For example:

   ```
   gfsh>backup disk-store --dir=/export/fileServerDirectory/gemfireBackupLocation
   ```

   The output will list information for each member that has successfully backed up disk stores. The tabular information will contain the member's name, its UUID, the directory backed up, and the host name of the member.

   Any online member that fails to complete its backup will leave a file named `INCOMPLETE_BACKUP` in its highest level backup directory. The existence of this file identifies that the backup file contains only a partial backup, and it cannot be used in a restore operation.

3. Validate the backup for later recovery use. On the command line, each backup can be checked with commands such as

   ```
   cd 2010-04-10-11-35/straw_14871_53406_34322/diskstores/ds1
   gfsh validate offline-disk-store --name=ds1 --disk-dirs=/home/dsmith/dir1
   ```

How to Do an Incremental Backup

An incremental backup contains items that have changed since a previous backup was made.

To do an incremental backup, specify the backup directory that the incremental backup will be based upon with the `--baseline-dir` argument. For example:

```
gfsh>backup disk-store --dir=/export/fileServerDirectory/gemfireBackupLocation
--baseline-dir=/export/fileServerDirectory/gemfireBackupLocation/2012-10-01-12-30
```

The output will appear the same as the output for a full online backup.

Any online member that fails to complete its incremental backup will leave a file named `INCOMPLETE_BACKUP` in its highest level backup directory. The existence of this file identifies that the backup file contains only a partial backup, and it cannot be used in a restore operation. The next time a backup is made, a full backup will be made.

What a Full Online Backup Saves

For each member with persistent data, a full backup includes the following:

- Disk store files for all members containing persistent region data.
- Files and directories specified in the `cache.xml` configuration file as `<backup>` elements. For example:

```
<backup>./systemConfig/gf.jar</backup>
<backup>/users/user/gfSystemInfo/myCustomerConfig.doc</backup>
```

- Deployed JAR files that were deployed using the gfsh deploy command.
- Configuration files from the member startup.
    - `gemfire.properties`, including the properties with which the member was started.
    - `cache.xml`, if used.

  These configuration files are not automatically restored, to avoid interfering with more recent configurations. In particular, if these are extracted from a primary `jar` file, copying the separate files into your working area can override the files in the `jar`. If you want to back up and restore these files, add them as custom `<backup>` elements.
- A restore script, called `restore.bat` on Windows, and called `restore.sh` on Linux. This script may later be used to do a restore. The script copies files back to their original locations.

What an Incremental Online Backup Saves

An incremental backup saves the difference between the last backup and the current data. An incremental backup copies only operations logs that are not already present in the baseline directories for each member. For incremental backups, the restore script contains explicit references to operation logs in one or more previously chained incremental backups. When the restore script is run from an incremental backup, it also restores the operation logs from previous incremental backups that are part of the backup chain.

If members are missing from the baseline directory because they were offline or did not exist at the time of the baseline backup, those members place full backups of all their files into the incremental backup directory.

Disk Store Backup Directory Structure and Contents

```
$ cd thebackupdir
$ ls -R
./2012-10-18-13-44-53:
dasmith_e6410_server1_8623_v1_33892 dasmith_e6410_server2_8940_v2_45565

./2012-10-18-13-44-53/dasmith_e6410_server1_8623_v1_33892:
config diskstores README.txt restore.sh user

./2012-10-18-13-44-53/dasmith_e6410_server1_8623_v1_33892/config:
cache.xml

./2012-10-18-13-44-53/dasmith_e6410_server1_8623_v1_33892/diskstores:
DEFAULT

./2012-10-18-13-44-53/dasmith_e6410_server1_8623_v1_33892/diskstores/DEFAULT:
dir0

./2012-10-18-13-44-53/dasmith_e6410_server1_8623_v1_33892/diskstores/DEFAULT/dir0:
BACKUPDEFAULT_1.crf BACKUPDEFAULT_1.drf BACKUPDEFAULT.if
```

```
./2012-10-18-13-44-53/dasmith_e6410_server1_8623_v1_33892/user:
```

Offline Members—Manual Catch-Up to an Online Backup

If you must have a member offline during an online backup, you can manually back up its disk stores. Bring this member's files into the online backup framework manually, and create a restore script by hand starting with a copy of another member's script:

1. Duplicate the directory structure of a backed up member for this member.
2. Rename directories as needed to reflect this member's particular backup, including disk store names.
3. Clear out all files other than the restore script.
4. Copy in this member's files.
5. Modify the restore script to work for this member.

Restore Using a Backup Made While the System Was Online

The `restore.sh` or `restore.bat` script copies files back to their original locations.

1. Restore your disk stores while cache members are offline and the system is down.
2. Look at each of the restore scripts to see where they will place the files and make sure the destination locations are ready. A restore script will refuse to copy over files with the same names.
3. Run each restore script on the host where the backup originated.

The restore copies these files back to their original location:

- Disk store files for all stores containing persistent region data.
- Any files or directories you have configured to be backed up in the `cache.xml` `<backup>` elements.

Cache and Region Snapshots

Snapshots allow you to save region data and reload it later. A typical use case is loading data from one environment into another, such as capturing data from a production system and moving it into a smaller QA or development system.

In effect, you can load data from one cluster into another cluster. Administrators export a snapshot of a region or an entire cache (multiple regions) and later import the snapshot into another region or cluster by using the `RegionSnapshotService` or `CacheSnapshotService` interface and the `Region.getSnapshotService` or `Cache.getSnapshotService` method.

The snapshot file is a binary file that contains all data from a particular region. The binary format contains serialized key/value pairs and supports PDX type registry to allow the deserialization of PDX data. The snapshot can be directly imported into a region or read entry-by-entry for further processing or transformation into other formats.

Note: The previous `Region.loadSnapshot` and `Region.saveSnapshot` APIs have been deprecated. Data written in this format is not compatible with the new APIs.

- **Usage and Performance Notes**

  Optimize the cache and region snapshot feature by understanding how it performs.

- **Exporting Cache and Region Snapshots**

  To save VMware GemFire cache or region data to a snapshot that you can later load into another cluster or region, use the `cache.getSnapshotService.save` API, `region.getSnapshotService.save` API, or the `gfsh` command-line interface (`export data`).

- **Importing Cache and Region Snapshots**

  To import a VMware GemFire cache or region data snapshot that you previously exported into another cluster or region, use the `cache.getSnapshotService.load` API, `region.getSnapshotService.load` API, or the `gfsh` command-line interface (`import data`).

- **Filtering Entries During Import or Export**

  You can customize your snapshot by filtering entries during the import or export of a region or a cache.

- **Reading Snapshots Programmatically**

  You can read a snapshot entry-by-entry for further processing or transformation into other formats.

Usage and Performance Notes

Optimize the cache and region snapshot feature by understanding how it performs.

Cache Consistency and Concurrent Operations

Importing and exporting region data is an administrative operation, and certain simultaneous runtime conditions can cause the import or export operation to fail such as when you are rebalancing partitioned region buckets or experience a network partition event. This behavior is expected, and you should retry the operation. Redoing an export overwrites an incomplete snapshot file, and redoing an import updates partially imported data.

The snapshot feature does not guarantee consistency. Concurrent cache operations during a snapshot import or export can cause data consistency issues. If snapshot consistency is important, we recommend that you take your application offline before export and import, to provide a quiet period ensures data consistency in your snapshot.

For example, modifications to region entries during an export can result in a snapshot that contains some but not all updates. If entries { A, B } are updated to { A', B'} during the export, the snapshot can contain { A, B' } depending on the write order. Also, modifications to region entries during an import can cause lost updates in the cache. If the region contains entries { A, B } and the snapshot contains { A', B' }, concurrent updates { A*, B* } can result in the region containing { A*, B' } after the import completes.

The default behavior is to perform all I/O operations on the node where the snapshot operations are invoked. This will involve either collecting or dispersing data over the network if the region is a partitioned region.

Performance Considerations

When using the data snapshot feature, be aware of the following performance considerations:

- Importing and exporting cache or region snapshots causes additional CPU and network load. You may need to increase CPU capacity or network bandwidth depending on your applications and infrastructure. In addition, if you export regions that have been configured to overflow to disk, you may require additional disk I/O to perform the export.
- When exporting partitioned region data, allocate additional heap memory so the member performing the export can buffer data gathered from other cache members. Allocate at least 10MB per member to your heap in addition to whatever configuration is necessary to support your application or cache.

Exporting Cache and Region Snapshots

To save VMware GemFire cache or region data to a snapshot that you can later load into another cluster or region, use the `cache.getSnapshotService.save` API, `region.getSnapshotService.save` API, or the `gfsh` command-line interface (`export data`).

If an error occurs during export, the export halts and the snapshot operation is canceled. Typical errors that halt an export include scenarios such as full disk, problems with file permissions, and network partitioning.

Exporting Cache Snapshots

When you export an entire cache, it exports all regions in the cache as individual snapshot files into a directory. If no directory is specified, the default is the current directory. A snapshot file is created for each region, and the export operation automatically names each snapshot filename using the following convention:

`snapshot-<region>[-<subregion>]*`

When the export operation writes the snapshot filename, it replaces each forward slash (‘/’) in the region path with a dash (‘-’).

Using Java API:

```
File mySnapshotDir = ...
Cache cache = ...

cache.getSnapshotService().save(mySnapshotDir, SnapshotFormat.GEMFIRE);
```

Optionally, you can set a filter on the snapshot entries during the export. See Filtering Entries During Import or Export for an example.

Exporting a Region Snapshot

You can also export a specific region using the API or gfsh commands below.

Note: In the case of non-persistent regions, the snapshot that you export contains both in-cache entries and entries that overflow to disk.

Java API:

```
File mySnapshot = ...
Region<String, MyObject> region = ...

region.getSnapshotService().save(mySnapshot, SnapshotFormat.GEMFIRE);
```

gfsh:

Open a gfsh prompt. After connecting to a VMware GemFire cluster, at the prompt type:

```
gfsh>export data --region=Region --file=FileName.gfd --member=MemberName
```

where *Region* corresponds to the name of the region that you want to export, *FileName* (must end in .gfd) corresponds to the name of the export file and *MemberName* corresponds to a member that hosts the region. For example:

```
gfsh>export data --region=region1 --file=region1_2012_10_10.gfd --member=server1
```

The snapshot file will be written on the remote member at the location specified by the `--file` argument. For example, in the example command above, the `region1_2012_10_10.gfd` file will be written in the working directory of `server1`. For more information on this command, see export data.

Export Example with Options

These examples show how to include the `parallel` option for exporting partitioned regions. Note that the `parallel` option takes a directory rather than a file; see export data for details.

Java API:

```
File mySnapshotDir = ...
Region<String, MyObject> region = ...

SnapshotOptions<Integer, MyObject> options =
 region.getSnapshotServive.createOptions().setParallelMode(true);
region.getSnapshotService().save(mySnapshotDir, SnapshotFormat.GEMFIRE, options);
```

gfsh:

The Java API example, above, accomplishes the same purpose as the following gfsh command:

```
gfsh>export data --parallel --region=region1 --dir=region1_2012_10_10 --member=server1
```

Importing Cache and Region Snapshots

To import a VMware GemFire cache or region data snapshot that you previously exported into another cluster or region, use the `cache.getSnapshotService.load` API, `region.getSnapshotService.load` API, or the `gfsh` command-line interface (`import data`).

Import Requirements

Before you import a region snapshot:

- Make sure the cache is configured correctly. Configure all registered PdxSerializers, DataSerializers, and Instantiators; create regions; and ensure the classpath contains any required classes.
- When you import a snapshot containing PDX types, you must wait until the exported type definitions are imported into the cache before inserting data that causes type conflicts. It is recommended that you wait for the import to complete before inserting data.

Import Limitations

During an import, the `CacheWriter` and `CacheListener` callbacks are not invoked.

If an error occurs during import, the import is halted and the region will contain some but not all snapshot data.

The state of a cache client is indeterminate after an import. It is likely that the data in the client's cache is inconsistent with the imported data. Take the client offline during the import and restart it after the import completes.

Importing Cache Snapshots

When you import a cache snapshot, the snapshot file is imported into the same region (match determined by name) that was used during snapshot export. When you import a cache, you import all snapshot files located within a directory into the cache. The API attempts to load all files in the specified directory.

Java API:

```
File mySnapshotDir = ...
Cache cache = ...

cache.getSnapshotService().load(mySnapshotDir, SnapshotFormat.GEMFIRE);
```

Importing a Region Snapshot

Java API:

```
File mySnapshot = ...
Region<String, MyObject> region = ...

region.getSnapshotService().load(mySnapshot, SnapshotFormat.GEMFIRE);
```

gfsh:

Open a gfsh prompt. After connecting to a VMware GemFire cluster, at the prompt type:

```
gfsh>import data --region=Region --file=FileName.gfd --member=MemberName
```

where *Region* corresponds to the name of the region that you want to import data into; *FileName* (must end in .gfd) corresponds to the name of the file to be imported; and *MemberName* corresponds to a member that hosts the region. For example:

```
gfsh>import data --region=region1 --file=region1_2012_10_10.gfd --member=server2
```

The snapshot file must already reside on the specified member at the location specified in the `--file` argument before import.

For more information on this command, see import data. For an example of how to invoke this command with additional options, see Export Example with Options.

Filtering Entries During Import or Export

You can customize your snapshot by filtering entries during the import or export of a region or a cache.

For example, use filters to limit the export of data to a certain date range. If you set up a filter on the import or export of a cache, the filter is applied to every single region in the cache.

The following example filters snapshot data by even numbered keys.

```
File mySnapshot = ...
Region<Integer, MyObject> region = ...

SnapshotFilter<Integer, MyObject> even = new SnapshotFilter<Integer, MyObject>() {
  @Override
  public boolean accept(Entry<Integer, MyObject> entry) {
    return entry.getKey() % 2 == 0;
  }
};

RegionSnapshotService<Integer, MyObject> snapsrv = region.getSnapshotService();
SnapshotOptions<Integer, MyObject> options = snapsrv.createOptions().setFilter(even);

// only save cache entries with an even key
snapsrv.save(mySnapshot, SnapshotFormat.GEMFIRE, options);
```

Reading Snapshots Programmatically

You can read a snapshot entry-by-entry for further processing or transformation into other formats.

The following is an example of a snapshot reader that processes entries from a previously generated snapshot file.

```
File mySnapshot = ...
SnapshotIterator<String, MyObject> iter = SnapshotReader.read(mySnapshot);
try {
  while (iter.hasNext()) {
    Entry<String, MyObject> entry = iter.next();

    String key = entry.getKey();
    MyObject value = entry.getValue();

    System.out.println(key + " = " + value);
  }
} finally {
  iter.close();
}
```

Region Compression

This section describes region compression, its benefits and usage.

One way to reduce memory consumption by VMware GemFire is to enable compression in your regions. VMware GemFire allows you to compress in-memory region values using pluggable compressors (compression codecs). VMware GemFire includes the Snappy compressor as the built-in compression codec; however, you can implement and specify a different compressor for each compressed region.

What Gets Compressed

When you enable compression in a region, all values stored in the region are compressed while in memory. Keys and indexes are not compressed. New values are compressed when put into the in-memory cache and all values are decompressed when being read from the cache. Values are not compressed when persisted to disk. Values are decompressed before being sent over the wire to other peer members or clients.

When compression is enabled, each value in the region is compressed, and each region entry is compressed as a single unit. It is not possible to compress individual fields of an entry.

You can have a mix of compressed and non-compressed regions in the same cache.

- **Guidelines on Using Compression**

  This topic describes factors to consider when deciding on whether to use compression.

- **How to Enable Compression in a Region**

  This topic describes how to enable compression on your region.

- **Working with Compressors**

  When using region compression, you can use the default Snappy compressor included with VMware GemFire or you can specify your own compressor.

- **Comparing Performance of Compressed and Non-Compressed Regions**

  The comparative performance of compressed regions versus non-compressed regions can vary depending on how the region is being used and whether the region is hosted in a memory-bound JVM.

Guidelines on Using Compression

This topic describes factors to consider when deciding on whether to use compression.

Review the following guidelines when deciding on whether or not to enable compression in your region:

- **Use compression when JVM memory usage is too high.** Compression allows you to store more region data in-memory and to reduce the number of expensive garbage collection cycles that prevent JVMs from running out of memory when memory usage is high.

To determine if JVM memory usage is high, examine the the following statistics:

- vmStats>freeMemory
- vmStats->maxMemory
- ConcurrentMarkSweep->collectionTime

If the amount of free memory regularly drops below 20% - 25% or the duration of the garbage collection cycles is generally on the high side, then the regions hosted on that JVM are good candidates for having compression enabled.

- **Consider the types and lengths of the fields in the region's entries.** Since compression is performed on each entry separately (and not on the region as a whole), consider the potential for duplicate data across a single entry. Duplicate bytes are compressed more easily. Also, since region entries are first serialized into a byte area before being compressed, how well the data might compress is determined by the number and length of duplicate bytes across the entire entry and not just a single field. Finally, the larger the entry the more likely compression will achieve good results as the potential for duplicate bytes, and a series of duplicate bytes, increases.
- **Consider the type of data you wish to compress.** The type of data stored has a significant impact on how well the data may compress. String data will generally compress better than numeric data simply because string bytes are far more likely to repeat; however, that may not always be the case. For example, a region entry that holds a couple of short, unique strings may not provide as much memory savings when compressed as another region entry that holds a large number of integer values. In short, when evaluating the potential gains of compressing a region, consider the likelihood of having duplicate bytes, and more importantly the length of a series of duplicate bytes, for a single, serialized region entry. In addition, data that has already been compressed, such as JPEG format files, can actually cause more memory to be used.
- **Compress if you are storing large text values.** Compression is beneficial if you are storing large text values (such as JSON or XML) or blobs in VMware GemFire that would benefit from compression.
- **Consider whether fields being queried against are indexed.** You can query against compressed regions; however, if the fields you are querying against have not been indexed, then the fields must be decompressed before they can be used for comparison. In short, you may incur some query performance costs when querying against non-indexed fields.
- **Objects stored in the compression region must be serializable.** Compression only operates on byte arrays, therefore objects being stored in a compressed region must be serializable and deserializable. The objects can either implement the Serializable interface or use one of the other VMware GemFire serialization mechanisms (such as PdxSerializable). Implementers should always be aware that when compression is enabled the instance of an object put into a region will not be the same instance when taken out. Therefore, transient attributes will lose their value when the containing object is put into and then taken out of a region.
- **Compressed regions will enable cloning by default.** Setting a compressor and then disabling cloning results in an exception. The options are incompatible because the process

of compressing/serializing and then decompressing/deserializing will result in a different instance of the object being created and that may be interpreted as cloning the object.

How to Enable Compression in a Region

This topic describes how to enable compression on your region.

To enable compression on your region, set the following region attribute in your cache.xml:

```
<?xml version="1.0" encoding= "UTF-8"?>
<cache xmlns="http://geode.apache.org/schema/cache"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://geode.apache.org/schema/cache http://geode.apache.org/s
chema/cache/cache-1.0.xsd"
    version="1.0” lock-lease="120"  lock-timeout= "60" search-timeout= "300"  is-serve
r= "true"  copy-on-read= "false" >
   <region name="compressedRegion" >
      <region-attributes data-policy="replicate" ... />
         <compressor>
             <class-name>org.apache.geode.compression.SnappyCompressor</class-name>
         </compressor>
        ...
      </region-attributes>
   </region>
</cache>
```

In the Compressor element, specify the class-name for your compressor implementation. This example specifies the Snappy compressor, which is bundled with VMware GemFire . You can also specify a custom compressor. See Working with Compressors for an example.

Compression can be enabled during region creation using gfsh or programmatically as well.

Using gfsh:

```
gfsh>create-region --name=”CompressedRegion” --compressor=”org.apache.geode.compressio
n.SnappyCompressor”;
```

API:

```
regionFactory.setCompressor(new SnappyCompressor());
```

or

```
regionFactory.setCompressor(SnappyCompressor.getDefaultInstance());
```

How to Check Whether Compression is Enabled

You can also check whether a region has compression enabled by querying which codec is being used. A null codec indicates that no compression is enabled for the region.

```
Region myRegion = cache.getRegion("myRegion");
Compressor compressor = myRegion.getAttributes().getCompressor();
```

Working with Compressors

When using region compression, you can use the default Snappy compressor included with VMware GemFire or you can specify your own compressor.

The compression API consists of a single interface that compression providers must implement. The default compressor (SnappyCompressor) is the single compression implementation that comes bundled with the product. Note that since the Compressor is stateless, there only needs to be a single instance in any JVM; however, multiple instances may be used without issue. The single, default instance of the SnappyCompressor may be retrieved with the `SnappyCompressor.getDefaultInstance()` static method.

Note: The Snappy codec included with VMware GemFire cannot be used with Solaris deployments. Snappy is only supported on Linux, Windows, and macOS deployments of VMware GemFire.

This example provides a custom Compressor implementation:

```
package com.mybiz.myproduct.compression;

import org.apache.geode.compression.Compressor;

public class LZWCompressor implements Compressor {
  private final LZWCodec lzwCodec = new LZWCodec();

  @Override
  public byte[] compress(byte[] input) {
         return lzwCodec.compress(input);
  }

  @Override
  public byte[] decompress(byte[] input) {
         return lzwCodec.decompress(input);
  }
}
```

To use the new custom compressor on a region:

1. Make sure that the new compressor package is available in the classpath of all JVMs that will host the region.
2. Configure the custom compressor for the region using any of the following mechanisms:

   Using gfsh:

   ```
   gfsh>create-region --name=”CompressedRegion” \
   --compressor=”com.mybiz.myproduct.compression.LZWCompressor”
   ```

   Using API:

   For example:

   ```
   regionFactory.setCompressor(new LZWCompressor());
   ```

   cache.xml:

   ```
   <region-attributes>
    <Compressor>
   ```

```
    <class-name>com.mybiz.myproduct.compression.LZWCompressor</class-name>
  </Compressor>
</region-attributes>
```

Changing the Compressor for an Already Compressed Region

You typically enable compression on a region at the time of region creation. You cannot modify the Compressor or disable compression for the region while the region is online.

However, if you need to change the compressor or disable compression, you can do so by performing the following steps:

1. Shut down the members hosting the region you wish to modify.
2. Modify the cache.xml file for the member either specifying a new compressor or removing the compressor attribute from the region.
3. Restart the member.

Comparing Performance of Compressed and Non-Compressed Regions

The comparative performance of compressed regions versus non-compressed regions can vary depending on how the region is being used and whether the region is hosted in a memory-bound JVM.

When considering the cost of enabling compression, you should consider the relative cost of reading and writing compressed data as well as the cost of compression as a percentage of the total time spent managing entries in a region. As a general rule, enabling compression on a region will add 30% - 60% more overhead for region create and update operations than for region get operations. Because of this, enabling compression will create more overhead on regions that are write heavy than on regions that are read heavy.

However, when attempting to evaluate the performance cost of enabling compression you should also consider the cost of compression relative to the overall cost of managing entries in a region. A region may be tuned in such a way that it is highly optimized for read and/or write performance. For example, a replicated region that does not save to disk will have much better read and write performance than a partitioned region that does save to disk. Enabling compression on a region that has been optimized for read and write performance will provide more noticeable results than using compression on regions that have not been optimized this way. More concretely, performance may degrade by several hundred percent on a read/write optimized region whereas it may only degrade by 5 to 10 percent on a non-optimized region.

A final note on performance relates to the cost when enabling compression on regions in a memory bound JVM. Enabling compression generally assumes that the enclosing JVM is memory bound and therefore spends a lot of time for garbage collection. In that case performance may improve by as much as several hundred percent as the JVM will be running far fewer garbage collection cycles and spending less time when running a cycle.

Monitoring Compression Performance

The following statistics provide monitoring for cache compression:

- `compressTime`
- `decompressTime`
- `compressions`
- `decompressions`
- `preCompressedBytes`
- `postCompressedBytes`

See Cache Performance (CachePerfStats) for statistic descriptions.

Network Partitioning

VMware GemFire architecture and management features help detect and resolve network partition problems.

- **How Network Partitioning Management Works**

  VMware GemFire handles network outages by using a weighting system to determine whether the remaining available members have a sufficient quorum to continue as a cluster.

- **Failure Detection and Membership Views**

  VMware GemFire uses failure detection to remove unresponsive members from membership views.

- **Membership Coordinators, Lead Members and Member Weighting**

  Network partition detection uses a designated membership coordinator and a weighting system that accounts for a lead member to determine whether a network partition has occurred.

- **Network Partitioning Scenarios**

  This topic describes network partitioning scenarios and what happens to the partitioned sides of the cluster.

- **Configure VMware GemFire to Handle Network Partitioning**

  This section lists the configuration steps for network partition detection.

- **Preventing Network Partitions**

  This section provides a short list of things you can do to prevent network partition from occurring.

How Network Partitioning Management Works

VMware GemFire handles network outages by using a weighting system to determine whether the remaining available members have a sufficient quorum to continue as a cluster.

Individual members are each assigned a weight, and the quorum is determined by comparing the total weight of currently responsive members to the previous total weight of responsive members.

Your cluster can split into separate running systems when members lose the ability to see each other. The typical cause of this problem is a failure in the network. When a partitioned system is detected, only one side of the system keeps running and the other side automatically shuts down.

The network partitioning detection feature is enabled by default with a true value for the `enable-network-partition-detection` property. See Configure VMware GemFire to Handle Network Partitioning for details. Quorum weight calculations are always performed and logged regardless of this configuration setting.

The overall process for detecting a network partition is as follows:

1. The cluster starts up. When you start up a cluster, start the locators first, start the cache servers second, and then start other members such as applications or processes that access cluster data.
2. After the members start up, the oldest member, typically a locator, assumes the role of the membership coordinator. Peer discovery occurs as members come up and members generate a membership discovery list for the cluster. Locators hand out the membership discovery list as each member process starts up. This list typically contains a hint on who the current membership coordinator is.
3. Members join and if necessary, depart the cluster:
    - Member processes make a request to the coordinator to join the cluster. If authenticated, the coordinator creates a new membership view, hands the new membership view to the new member, and begins the process of sending the new membership view (to add the new member or members) by sending out a view preparation message to existing members in the view.
    - While members are joining the system, it is possible that members are also leaving or being removed through the normal failure detection process. Failure detection removes unresponsive or slow members. See Managing Slow Receivers and Failure Detection and Membership Views for descriptions of the failure detection process. If a new membership view is sent out that includes one or more failed processes, the coordinator will log the new weight calculations. At any point, if quorum loss is detected due to unresponsive processes, the coordinator will also log a severe level message to identify the failed processes:

      ```
      Possible loss of quorum detected due to loss of {0} cache processes: {1}
      ```

      where {0} is the number of processes that failed and {1} lists the processes.
4. Whenever the coordinator is alerted of a membership change (a member either joins or leaves the cluster), the coordinator generates a new membership view. The membership view is generated by a two-phase protocol:
    1. In the first phase, the membership coordinator sends out a view preparation message to all members and waits 12 seconds for a view preparation ack return message from each member. If the coordinator does not receive an ack message from a member within 12 seconds, the coordinator attempts to connect to the member's failure-detection socket. If the coordinator cannot connect to the member's failure-detection socket, the coordinator declares the member dead and starts the membership view protocol again from the beginning.

    2. In the second phase, the coordinator sends out the new membership view to all members that acknowledged the view preparation message or passed the connection test.

5. Each time the membership coordinator sends a view, each member calculates the total weight of members in the current membership view and compares it to the total weight of the previous membership view. Some conditions to note:
    - When the first membership view is sent out, there are no accumulated losses. The first view only has additions.
    - A new coordinator may have a stale view of membership if it did not see the last membership view sent by the previous (failed) coordinator. If new members were added during that failure, then the new members may be ignored when the first new view is sent out.
    - If members were removed during the fail over to the new coordinator, then the new coordinator will have to determine these losses during the view preparation step.
6. With a default value of `enable-network-partition-detection`, any member that detects that the total membership weight has dropped below 51% within a single membership view change (loss of quorum) declares a network partition event. The coordinator sends a network-partitioned-detected UDP message to all members (even to the non-responsive ones) and then closes the cluster with a `ForcedDisconnectException`. If a member fails to receive the message before the coordinator closes the system, the member is responsible for detecting the event on its own.

The presumption is that when a network partition is declared, the members that comprise a quorum will continue operations. The surviving members elect a new coordinator, designate a lead member, and so on.

Failure Detection and Membership Views

VMware GemFire uses failure detection to remove unresponsive members from membership views.

Failure Detection

Network partitioning has a failure detection protocol that is not subject to hanging when NICs or machines fail. Failure detection has each member observe messages from the peer to its right within the membership view (see “Membership Views” below for the view layout). A member that suspects the failure of its peer to the right sends a datagram heartbeat request to the suspect member. With no response from the suspect member, the suspicious member broadcasts a `SuspectMembersMessage` datagram message to all other members. The coordinator attempts to connect to the suspect member. If the connection attempt is unsuccessful, the suspect member is removed from the membership view. The suspect member is sent a message to disconnect from the cluster and close the cache. In parallel to the receipt of the `SuspectMembersMessage`, a distributed algorithm promotes the leftmost member within the view to act as the coordinator, if the coordinator is the suspect member.

Failure detection processing is also initiated on a member if the `gemfire.properties` `ack-wait-threshold` elapses before receiving a response to a message, if a TCP/IP connection cannot be made to the member for peer-to-peer (P2P) messaging, and if no other traffic is detected from the member.

Note: The TCP connection ping is not used for connection keep alive purposes; it is only used to detect failed members. See TCP/IP KeepAlive Configuration for TCP keep alive configuration.

If a new membership view is sent out that includes one or more failed members, the coordinator will log new quorum weight calculations. At any point, if quorum loss is detected due to unresponsive processes, the coordinator will also log a severe level message to identify the failed members:

```
Possible loss of quorum detected due to loss of {0} cache processes: {1}
```

in which {0} is the number of processes that failed and {1} lists the members (cache processes).

Membership Views

The following is a sample membership view:

```
[info 2012/01/06 11:44:08.164 PST bridgegemfire1 <UDP Incoming Message Handler> tid=0x
1f]
Membership: received new view  [ent(5767)<v0>:8700|16] [ent(5767)<v0>:8700/44876,
ent(5829)<v1>:48034/55334, ent(5875)<v2>:4738/54595, ent(5822)<v5>:49380/39564,
ent(8788)<v7>:24136/53525]
```

The components of the membership view are as follows:

- The first part of the view (`[ent(5767)<v0>:8700|16]` in the example above) corresponds to the view ID. It identifies:
  - the address and processId of the membership coordinator: `ent(5767)` in example above.
  - the view-number (`<vXX>`) of the membership view that the member first appeared in: `<v0>` in example above.
  - membership-port of the membership coordinator: `8700` in the example above.
  - view-number: `16` in the example above
- The second part of the view lists all of the member processes in the current view. `[ent(5767)<v0>:8700/44876, ent(5829)<v1>:48034/55334, ent(5875)<v2>:4738/54595, ent(5822)<v5>:49380/39564, ent(8788)<v7>:24136/53525]` in the example above.
- The overall format of each listed member is:`Address(processId)<vXX>:membership-port/distribution port`. The membership coordinator is almost always the first member in the view and the rest are ordered by age.
- The membership-port is the JGroups TCP UDP port that it uses to send datagrams. The distribution-port is the TCP/IP port that is used for cache messaging.
- Each member watches the member to its right for failure detection purposes.

Membership Coordinators, Lead Members and Member Weighting

Network partition detection uses a designated membership coordinator and a weighting system that accounts for a lead member to determine whether a network partition has occurred.

Membership Coordinators and Lead Members

The membership coordinator is a member that manages entry and exit of other members of the cluster. With network partition detection enabled, the coordinator can be any VMware GemFire member but locators are preferred. In a locator-based system, if all locators are in the reconnecting state, the system continues to function, but new members are not able to join until a locator has successfully reconnected. After a locator has reconnected, the reconnected locator will take over the role of coordinator.

When a coordinator is shutting down, it sends out a view that removes itself from the list and the other members must determine who the new coordinator is.

The lead member is determined by the coordinator. Any member that has enabled network partition detection, is not hosting a locator, and is not an administrator interface-only member is eligible to be designated as the lead member by the coordinator. The coordinator chooses the longest-lived member that fits the criteria.

The purpose of the lead member role is to provide extra weight. It does not perform any specific functionality.

Member Weighting System

By default, individual members are assigned the following weights:

- Each member has a weight of 10 except the lead member.
- The lead member is assigned a weight of 15.
- Locators have a weight of 3.

You can modify the default weights for specific members by defining the `gemfire.member-weight` system property upon startup.

The weights of members prior to the view change are added together and compared to the weight of lost members. Lost members are considered members that were removed between the last view and the completed send of the view preparation message. If membership is reduced by a certain percentage within a single membership view change, a network partition is declared.

The loss percentage threshold is 51 (meaning 51%). Note that the percentage calculation uses standard rounding. Therefore, a value of 50.51 is rounded to 51. If the rounded loss percentage is equal to or greater than 51%, the membership coordinator initiates shut down.

Sample Member Weight Calculations

This section provides some example calculations.

Example 1: Cluster with 12 members. 2 locators, 10 cache servers (one cache server is designated as lead member.) View total weight equals 111.

- 4 cache servers become unreachable. Total membership weight loss is 40 (36%). Since 36% is under the 51% threshold for loss, the cluster stays up.
- 1 locator and 4 cache servers (including the lead member) become unreachable. Membership weight loss equals 48 (43%). Since 43% is under the 51% threshold for loss, the cluster stays up.
- 5 cache servers (not including the lead member) and both locators become unreachable. Membership weight loss equals 56 (49%). Since 49% is under the 51% threshold for loss, the cluster stays up.
- 5 cache servers (including the lead member) and 1 locator become unreachable. Membership weight loss equals 58 (52%). Since 52% is greater than the 51% threshold, the coordinator initiates shutdown.
- 6 cache servers (not including the lead member) and both locators become unreachable. Membership weight loss equals 66 (59%). Since 59% is greater than the 51% threshold, the newly elected coordinator (a cache server since no locators remain) will initiate shutdown.

Example 2: Cluster with 4 members. 2 cache servers (1 cache server is designated lead member), 2 locators. View total weight is 31.

- Cache server designated as lead member becomes unreachable. Membership weight loss equals 15 or 48%. Cluster stays up.
- Cache server designated as lead member and 1 locator become unreachable. Member weight loss equals 18 or 58%. Membership coordinator initiates shutdown. If the locator that became unreachable was the membership coordinator, the other locator is elected coordinator and then initiates shutdown.

Even if network partitioning is not enabled, if quorum loss is detected due to unresponsive processes, the locator will also log a severe level message to identify the failed processes:

```
Possible loss of quorum detected due to loss of {0} cache processes: {1}
```

where {0} is the number of processes that failed and {1} lists the processes.

Enabling network partition detection allows only one subgroup to survive a split. The rest of the system is disconnected and the caches are closed.

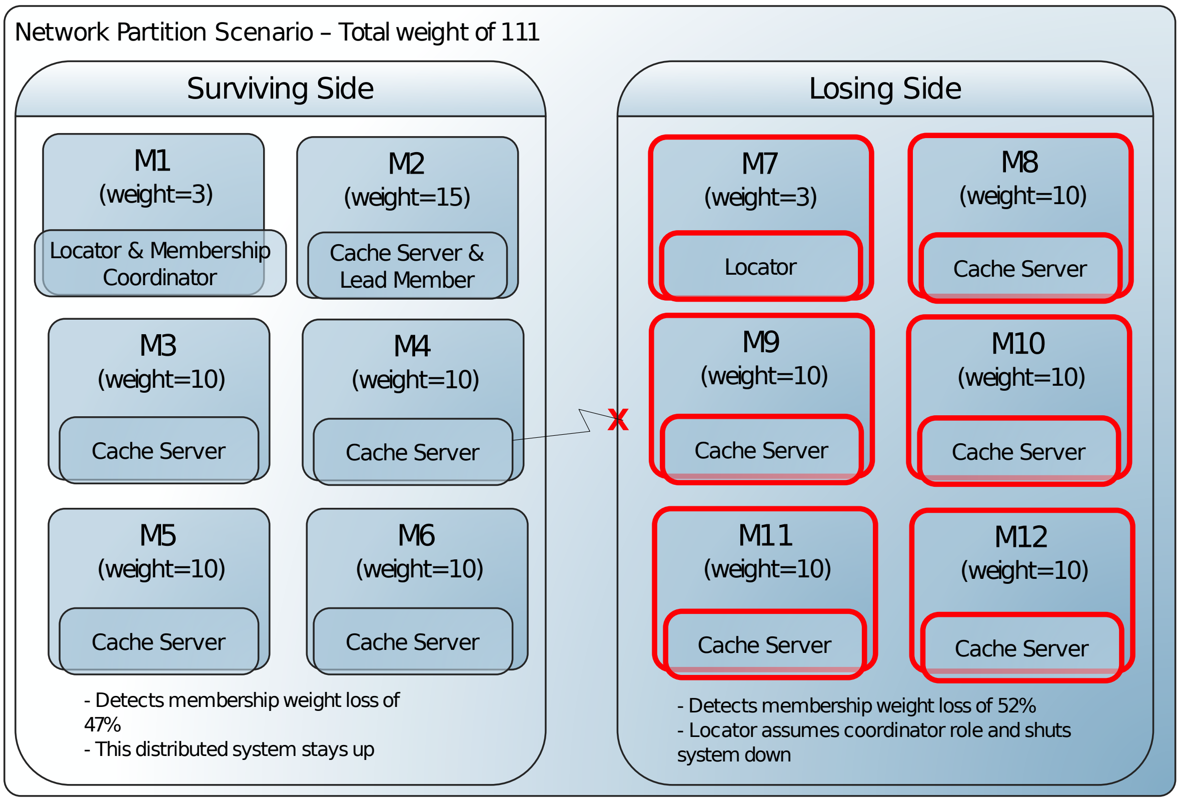
When a shutdown occurs, the members that are shut down will log the following alert message:

```
Exiting due to possible network partition event due to loss of {0} cache processes:
{1}
```

where `{0}` is the count of lost members and `{1}` is the list of lost member IDs.

Network Partitioning Scenarios

This topic describes network partitioning scenarios and what happens to the partitioned sides of the cluster.



What the Losing Side Does

In a network partitioning scenario, the “losing side” constitutes the cluster partition where the membership coordinator has detected that there is an insufficient quorum of members to continue.

The membership coordinator calculates membership weight change after sending out its view preparation message. If a quorum of members does not remain after the view preparation phase, the coordinator on the “losing side” declares a network partition event and sends a network-partition-detected UDP message to the members. The coordinator then closes its cluster with a `ForcedDisconnectException`. If a member fails to receive the message before the coordinator closes the connection, it is responsible for detecting the event on its own.

When the losing side discovers that a network partition event has occurred, all peer members receive a `RegionDestroyedException` with `Operation`: `FORCED_DISCONNECT`.

If a `CacheListener` is installed, the `afterRegionDestroy` callback is invoked with a `RegionDestroyedEvent`, as shown in this example logged by the losing side’s callback. The peer member process IDs are 14291 (lead member) and 14296, and the locator is 14289.

```
[info 2008/05/01 11:14:51.853 PDT <CloserThread> tid=0x4a]
Invoked splitBrain.SBListener: afterRegionDestroy in client1 whereIWasRegistered: 1429
1
event.isReinitializing(): false
event.getDistributedMember(): thor(14291):40440/34132
event.getCallbackArgument(): null
event.getRegion(): /TestRegion
event.isOriginRemote(): false
Operation: FORCED_DISCONNECT
```

```
Operation.isDistributed(): false
Operation.isExpiration(): false
```

Peers still actively performing operations on the cache may see `ShutdownException`s or `CacheClosedException`s with `Caused by: ForcedDisconnectException`.

What Isolated Members Do

When a member is isolated from all locators, it is unable to receive membership view changes. It can't know if the current coordinator is present or, if it has left, whether there are other members available to take over that role. In this condition, a member will eventually detect the loss of all other members and will use the loss threshold to determine whether it should shut itself down. In the case of a cluster with 2 locators and 2 cache servers, the loss of communication with the non-lead cache server plus both locators would result in this situation and the remaining cache server would eventually shut itself down.

Configure VMware GemFire to Handle Network Partitioning

This section lists configuration considerations relating to network partition detection.

The system uses a combination of member coordinators and system members, designated as lead members, to detect and resolve network partitioning problems.

- Network partition detection works in all environments. Using multiple locators mitigates the effect of network partitioning. See Configuring Peer-to-Peer Discovery.
- Network partition detection is enabled by default. The default setting in the `gemfire.properties` file is

  ```
  enable-network-partition-detection=true
  ```

  Processes that do not have network partition detection enabled are not eligible to be the lead member, so their failure will not trigger declaration of a network partition.

  All system members should have the same setting for `enable-network-partition-detection`. If they do not, the system throws a `GemFireConfigException` upon startup.
- The property `enable-network-partition-detection` must be true if you are using either partitioned or persistent regions. If you create a persistent region and `enable-network-partition-detection` to set to false, you will receive the following warning message:

  ```
  Creating persistent region {0}, but enable-network-partition-detection is set t
  o false.
        Running with network partition detection disabled can lead to an unrecove
  rable system in the
        event of a network split."
  ```
- Configure regions you want to protect from network partitioning with a scope setting of `DISTRIBUTED_ACK` or `GLOBAL`. Do not use `DISTRIBUTED_NO_ACK` scope. This prevents operations from being performed throughout the cluster before a network partition is

detected. **Note:** VMware GemFire issues an alert if it detects `DISTRIBUTED_NO_ACK` regions when network partition detection is enabled:

```
Region {0} is being created with scope {1} but enable-network-partition-detecti
on is enabled in the distributed system.
This can lead to cache inconsistencies if there is a network failure.
```

- These other configuration parameters affect or interact with network partitioning detection. Check whether they are appropriate for your installation and modify as needed.
  - If you have network partition detection enabled, the threshold percentage value for allowed membership weight loss is automatically configured to 51. You cannot modify this value. **Note:** The weight loss calculation uses round to nearest. Therefore, a value of 50.51 is rounded to 51 and will cause a network partition.
  - Failure detection is initiated if a member's `ack-wait-threshold` (default is 15 seconds) and `ack-severe-alert-threshold` (15 seconds) properties elapse before receiving a response to a message. If you modify the `ack-wait-threshold` configuration value, you should modify `ack-severe-alert-threshold` to match the other configuration value.
  - If the system has clients connecting to it, the clients' `cache.xml` pool `read-timeout` should be set to at least three times the `member-timeout` setting in the server's `gemfire.properties` file. The default pool `read-timeout` setting is 10000 milliseconds.
  - You can adjust the default weights of members by specifying the system property `gemfire.member-weight` upon startup. For example, if you have some VMs that host a needed service, you could assign them a higher weight upon startup.
- By default, members that are forced out of the cluster by a network partition event will automatically restart and attempt to reconnect. Data members will attempt to reinitialize the cache. See Handling Forced Cache Disconnection Using Autoreconnect.

Preventing Network Partitions

This section provides a short list of things you can do to prevent a network partition from occurring.

To avoid a network partition:

- Use NIC teaming for redundant connectivity. See http://www.cisco.com/en/US/docs/solutions/Enterprise/Data_Center/vmware/VMware.html#wp696452 for more information.
- It is best if all servers share a common network switch. Having multiple network switches increases the possibility of a network partition occurring. If multiple switches must be used, redundant routing paths should be available, if possible. The weight of members sharing a switch in a multi-switch configuration will determine which partition survives if there is an inter-switch failure.
- In terms of VMware GemFire configuration, consider the weighting of members. For example, you could assign important processes a higher weight.

Security

The security framework permits authentication of connecting components and authorization of operations for all communicating components of the cluster.

- **Security Implementation Introduction and Overview**

  Encryption, SSL secure communication, authentication, and authorization help to secure the cluster.

- **Security Detail Considerations**

  This section gathers discrete details in one convenient location to better help you assess and configure the security of your environment.

- **Enable Security with Property Definitions**

- **Authentication**

  A cluster using authentication bars malicious peers or clients, and deters inadvertent access to its cache.

- **Authorization**

  Client operations on a cache server can be restricted or completely blocked based on the roles and permissions assigned to the credentials submitted by the client.

- **Post Processing of Region Data**

- **SSL**

  SSL protects your data in transit between applications.

Security Implementation Introduction and Overview

Security Features

Encryption, SSL secure communication, authentication, and authorization features help to secure the cluster.

Security features include:

- **A single security interface for all components**. The single authentication and authorization mechanism simplifies the security implementation. It views and interacts with all components in a consistent manner.

- **System-wide role-based access control**. Roles regiment authorized operations requested by the various components.

- **SSL communication**. Allows configuration of connections to be SSL-based, rather than plain socket connections. You can enable SSL separately for peer-to-peer, client, JMX, gateway senders and receivers, and HTTP connections.

- **Post processing of region data**. Return values for operations that return region values may be formatted.

Overview

An authentication and authorization mechanism forms the core of the internal security of the cluster. Communications may be further protected by enabling SSL for data in transit.

Authentication verifies the identity of communicating components, leading to control over participation. The variety of participants include peer members, servers, clients, originators of JMX operations, Pulse, gateway senders and receivers representing WAN members of the system, and commands arriving from `gfsh` on behalf of system users or administrators.

Connection requests trigger the invocation of an authentication callback. This special-purpose callback is written as part of the application, and it attempts to authenticate the requester by whatever algorithm it chooses. The result is either a returned principal representing the requester's authenticated identity or an exception indicating that the requester has not been authenticated. The principal becomes part of any request for operations, which go through the authorization process.

Given authentication, isolation and access to cache data and system state can be further protected by implementing the authorization mechanism, also implemented as a special-purpose callback as part of the application. For example, the protection may be to permit only certain system administrators to start and stop servers. The authority to do this needs to be limited to specific verified accounts, preventing those without the authorization. An implementation of the authorization callback will require that an authenticate identity accompanies all requests to the system, and that the system maintains a representation of which identities are permitted to complete which actions or cache commands.

Security Detail Considerations

This section gathers discrete details in one convenient location to better help you assess and configure the security of your environment.

- **External Interfaces, Ports, and Services**

  VMware GemFire processes use either UDP or TCP/IP ports to communicate with other processes or clients.

- **Resources That Must Be Protected**

  Certain VMware GemFire configuration files should be readable and writeable *only* by the dedicated user who runs servers.

- **Log File Locations**

  By default, the log files are located in the working directory used when you started the corresponding processes.

- **Where to Place Security Configuration Settings**

External Interfaces, Ports, and Services

VMware GemFire processes use either UDP or TCP/IP ports to communicate with other processes or clients.

For example:

- Members can use multicast to communicate with peer members. You specify multicast addresses and multicast ports in your `gemfire.properties` file or as parameters on the command-line when starting the members using `gfsh`.

- Clients connect to a locator to discover cache servers.
- JMX clients (such as `gfsh` and JConsole) can connect to JMX Managers and other manageable members on the pre-defined RMI port 1099. You can configure a different port if necessary.
- Each gateway receiver usually has a port range where it listens for incoming communication.

See Firewalls and Ports for the complete list of ports used by VMware GemFire, their default values, and how to configure them if you do not want to use the default value.

VMware GemFire does not have any external interfaces or services that need to be enabled or opened.

Resources That Must Be Protected

These configuration files should be readable and writeable *only* by the dedicated user who runs servers:

- `gemfire.properties`
- `cache.xml`
- `gfsecurity.properties` A default `gfsecurity.properties` is not provided in the `defaultConfigs` directory. If you choose to use this properties file, you must create it manually. A clear text user name and associated clear text password may be in this file for authentication purposes. The file system's access rights are relied upon to protect this sensitive information.

The default location of the `gemfire.properties` and `cache.xml` configuration files is the `defaultConfigs` child directory of the main installation directory.

Log File Locations

By default, the log files are located in the working directory used when you started the corresponding processes.

For VMware GemFire members (locators and cache servers), you can also specify a custom working directory location when you start each process. See Logging for more details.

The log files are as follows:

- `locator-name.log`: Contains logging information for the locator process.
- `server-name.log`: Contains logging information for a cache server process.
- `gfsh-%u_%g.log`: Contains logging information of an individual `gfsh` environment and session.

  Note: By default, `gfsh` session logging is disabled. To enable `gfsh` logging, you must set the Java system property `-Dgfsh. log-level=desired_log_level`. See Configuring the gfsh Environment for more information.

These log files should be readable and writable *only* by the dedicated user who runs the servers.

Where to Place Security Configuration Settings

Any security-related (properties that begin with `security-*`) configuration properties that are normally configured in `gemfire.properties` can be moved to a separate `gfsecurity.properties` file. Placing these configuration settings in a separate file allows you to restrict access to security configuration data. This way, you can still allow read or write access for your `gemfire.properties` file.

Upon startup, VMware GemFire processes will look for the `gfsecurity.properties` file in the following locations in order:

- current working directory
- user’s home directory
- classpath

If any password-related security properties are listed in the file but have a blank value, the process will prompt the user to enter a password upon startup.

Enable Security with Property Definitions

security-manager Property

The authentication callback and the authorization callback that implement the `SecurityManager` interface are specified with the `security-manager` property. When this property is defined, authentication and authorization are enabled. The definition of the `security-manager` property is the fully qualified name of the class that implements the `SecurityManager` interface. For example:

```
security-manager = com.example.security.MySecurityManager
```

Apply security-manager to All Members

To ensure that the `security-manager` property is applied consistently across a cluster, follow these guidelines:

- Specify the `security-manager` property in a properties file, such as `gemfire.properties`, **not** in a cluster configuration file (such as `cluster.properties`).
- Specify the properties file when you start the first locator for the cluster.

Is Cluster Management Enabled?

The next steps in applying the `security-manager` property across the cluster depend on whether cluster management is enabled. Cluster management is enabled when two conditions are met:

- Every locator in the cluster sets `--enable-cluster-configuration=true`.
- Every server in the cluster sets `--use-cluster-configuration=true`.

These are the default settings, so unless you have changed them, cluster management is probably enabled for your system, but be sure and confirm before proceeding. Some systems that implement cluster management for most members might include a few servers that do not participate (for which `--use-cluster-configuration=false`). See Using the Cluster Configuration Service for details.

Apply security-manager to Non-participating Servers

- **If cluster management is enabled (the default),** the locator will propagate the `security-manager` setting to all members (locators and servers) that are subsequently started.
- **If cluster management is enabled but some servers do not participate in cluster management** (that is, servers for which `--use-cluster-configuration=false`), you must specify the `security-manager` property for those non-participating servers. Make sure its value is exactly identical to that specified for the first locator.
- **If cluster management is not enabled,** you must specify the `security-manager` property for all servers. Make sure its value is exactly identical to that specified for the first locator.

Callbacks

All components of the system invoke the same callbacks. Here are descriptions of the components and the connections that they make with the system.

- A client connects with a server and makes operation requests of that server. The callbacks invoked are those defined by the `SecurityManager` interface for that server.
- A server connects with a locator, invoking the `authenticate` callback defined for that locator.
- Components communicating with a locator's JMX manager connect and make operation requests of the locator. The callbacks invoked are those defined by the `SecurityManager` interface for that locator. Both `gfsh` and `Pulse` use this form of communication.
- Applications communicating via the REST API make of a server invoke security callbacks upon connection and operation requests.
- Requests that a gateway sender makes of a locator invoke security callbacks defined for that locator.

security-post-processor Property

The `PostProcessor` interface allows the definition of a set of callbacks that are invoked after operations that get data, but before the data is returned. This permits the callback to intervene and format the data that is to be returned. The callbacks do not modify the region data, only the data to be returned.

Enable the post processing of data by defining the `security-post-processor` property with the path to the definition of the interface. For example,

```
security-post-processor = com.example.security.MySecurityPostProcessing
```

Authentication

Authentication verifies the identities of components within the cluster such as peers, clients, and those connecting to a JMX manager.

- **Implementing Authentication**

  All components of the cluster authenticate the same way, through a custom-written method.

- **Authentication Example**

  The example demonstrates the basics of an implementation of the `SecurityManager.authenticate` method.

Implementing Authentication

Authentication lends a measure of security to a cluster by verifying the identity of components as they connect to the system. All components use the same authentication mechanism.

How Authentication Works

When a component initiates a connection to the cluster, the `SecurityManager.authenticate` method is invoked. The component provides its credentials in the form of properties as a parameter to the `authenticate` method. The credential is presumed to be the two properties `security-username` and `security-password`. The `authenticate` method is expected to either return an object representing a principal or throw an `AuthenticationFailedException`.

A well-designed `authenticate` method will have a set of known user and password pairs that can be compared to the credential presented or will have a way of obtaining those pairs.

How a Server Sets Its Credential

In order to connect with a locator that does authentication, a server will need to set its credential, composed of the two properties `security-username` and `security-password`. Choose one of these two ways to accomplish this:

- Set `security-username` and `security-password` in the server's `gfsecurity.properties` file that will be read upon server start up, as in the example

  ```
  security-username=admin
  security-password=xyz1234
  ```

  The user name and password are stored in cleartext, so the `gfsecurity.properties` file must be protected by restricting access with file system permissions.

- Implement `AuthInitialize` interface for the server.
  - Set the property `security-peer-auth-init`, so that an object of the class that implements the `AuthInitialize` interface will be instantiated. Set the property to one of these two values:
    - Set property `security-peer-auth-init` to the fully-qualified class name that implements the `AuthInitialize` interface as in the example

    ```
    security-peer-auth-init=com.example.security.ServerAuthenticate
    ```

        - Set property `security-peer-auth-init` to the fully-qualified method name of a method that instantiates an object of the class that implements the `AuthInitialize` interface as in the example

        ```
        security-peer-auth-init=com.example.security.ServerAuthenticate.create
        ```

    - Implement the `getCredentials` method within the `AuthInitialize` interface to acquire values for the `security-username` and `security-password` properties in whatever way it wishes. It might look up values in a database or another external resource.

Gateway senders and receivers communicate as a component of their server member. Therefore, the credential of the server become those of the gateway sender or receiver.

How a Client Cache Sets Its Credential

In order to connect with a locator or a server that does authentication, a client will need to set its credential, composed of the two properties `security-username` and `security-password`. Choose one of these two ways to accomplish this:

- Set the `security-username` and `security-password` properties for the client using the API:

  ```
  Properties properties = new Properties();
  properties.setProperty("security-username", "exampleuser23");
  properties.setProperty("security-password", "xyz1234");
  ClientCache cache = new ClientCacheFactory(properties).create();
  ```

  Take care that credentials set in this manner are not accessible to observers of the code.

- Implement `AuthInitialize` interface for the client.
    - Set the property `security-client-auth-init`, so that an object of the class that implements the `AuthInitialize` interface will be instantiated. Set the property to one of these two values:
        - Set property `security-client-auth-init` to the fully-qualified class name that implements the `AuthInitialize` interface:

        ```
        security-client-auth-init=com.example.security.ClientAuthInitialize
        ```

        - Set property `security-client-auth-init` to the fully-qualified name of a static method that instantiates an object of the class that implements the `AuthInitialize` interface:

        ```
        security-client-auth-init=com.example.security.ClientAuthInitialize.creat
        e
        ```

    - Implement the `getCredentials` method of the `AuthInitialize` interface for the client. The implementation of `getCredentials` acquires values for the `security-username` and `security-password` properties in whatever way it wishes. It might look up values in a database or another external resource, or it might prompt for values.

How Other Components Set Their Credentials

`gfsh` prompts for the user name and password upon invocation of a`gfsh connect` command.

Pulse prompts for the user name and password upon start up.

Due to the stateless nature of the REST API, a web application or other component that speaks to a server or locator via the REST API goes through authentication on each request. The header of the request needs to include attributes that define values for `security-username` and `security-password`.

Implement SecurityManager Interface

Complete these items to implement authentication done by either a locator or a server.

- Decide upon an authentication algorithm. The Authentication Example stores a set of user name and password pairs that represent the identities of components that will connect to the system. This simplistic algorithm returns the user name as a principal if the user name and password passed to the `authenticate` method are a match for one of the stored pairs.
- Define the `security-manager` property. See Enable Security with Property Definitions for details about this property.
- Implement the `authenticate` method of the `SecurityManager` interface.
- Define any extra resources that the implemented authentication algorithm needs in order to make a decision.

Authentication Example

This example demonstrates the basics of an implementation of the `SecurityManager.authenticate` method. The remainder of the example may be found in the VMware GemFire source code in the `geode-core/src/main/java/org/apache/geode/examples/security` directory.

Of course, the security implementation of every installation is unique, so this example cannot be used in a production environment. Its use of the user name as a returned principal upon successful authentication is a particularly poor design choice, as any attacker that discovers the implementation can potentially spoof the system.

This example assumes that a set of user name and password pairs representing users that may be successfully authenticated has been read into a data structure upon initialization. Any component that presents the correct password for a user name successfully authenticates, and its identity is verified as that user. Therefore, the implementation of the `authenticate` method checks that the user name provided within the `credentials` parameter is in its data structure. If the user name is present, then the password provided within the `credentials` parameter is compared to the data structure's known password for that user name. Upon a match, the authentication is successful.

```
public Object authenticate(final Properties credentials)
         throws AuthenticationFailedException {
    String user = credentials.getProperty(ResourceConstants.USER_NAME);
    String password = credentials.getProperty(ResourceConstants.PASSWORD);

    User userObj = this.userNameToUser.get(user);
    if (userObj == null) {
        throw new AuthenticationFailedException(
                    "SampleSecurityManager: wrong username/password");
    }

    if (user != null
        && !userObj.password.equals(password)
        && !"".equals(user)) {
        throw new AuthenticationFailedException(
                    "SampleSecurityManager: wrong username/password");
    }

    return user;
}
```

Authorization

Cluster and cache operations can be restricted, intercepted and modifed, or completely blocked based on configured access rights set for the various cluster components.

- **Implementing Authorization**

  To use authorization for client/server systems, your client connections must be authenticated by their servers.

- **Method Invocation Authorizers**

  Authorizers used during query execution, how to configure them and how to implement your own.

- **Authorization Examples**

  This topic discusses the authorization examples provided in the product under `geode-core/src/main/java/org/apache/geode/examples/security`.

Implementing Authorization

How Authorization Works

When a component requests an operation, the `SecurityManager.authorize` method is invoked. It is passed the principal of the operation's requester and a `ResourcePermission`, which describes the operation requested.

The implementation of the `SecurityManager.authorize` method makes a decision as to whether or not the principal will be granted permission to carry out the operation. It returns a boolean in which a return value of `true` permits the operation, and a return value of `false` prevents the operation.

A well-designed `authorize` method will have or will have a way of obtaining a mapping of principals to the operations (in the form of resource permissions) that they are permitted to do.

Resource Permissions

All operations are described by an instance of the `ResourcePermission` class. A permission contains the `Resource` data member, which classifies whether the operation as working on

- cache data; value is `DATA`

- the cluster; value is `CLUSTER`

A permission also contains the `Operation` data member, which classifies whether the operation as

- reading; value is `READ`
- changing information; value is `WRITE`
- making administrative changes; value is `MANAGE`

The operations are not hierarchical; `MANAGE` does not imply `WRITE`, and `WRITE` does not imply `READ`.

Some `DATA` operations further specify a region name in the permission. This permits restricting operations on that region to only those authorized principals. And within a region, some operations may specify a key. This permits restricting operations on that key within that region to only those authorized principals.

Some `CLUSTER` operations further specify a finer-grained target for the operation. Specify the target with a string value of:

- `DISK` to target operations that write to a disk store
- `GATEWAY` to target operations that manage gateway senders and receivers
- `QUERY` to target operations that manage both indexes and continuous queries
- `DEPLOY` to target operations that deploy code to servers
- `LUCENE` to target Lucene index operations

This table classifies the permissions assigned for operations common to a Client-Server interaction.

Client Operation	Assigned `ResourcePermission`
get function attribute	CLUSTER:READ
create region	DATA:MANAGE
destroy region	DATA:MANAGE
Region.Keyset	DATA:READ:RegionName
Region.query	DATA:READ:RegionName
Region.getAll	DATA:READ:RegionName
Region.getAll with a list of keys	DATA:READ:RegionName:Key
Region.getEntry	DATA:READ:RegionName
Region.containsKeyOnServer(key)	DATA:READ:RegionName:Key
Region.get(key)	DATA:READ:RegionName:Key
Region.registerInterest(key)	DATA:READ:RegionName:Key
Region.registerInterest(regex)	DATA:READ:RegionName
Region.unregisterInterest(key)	DATA:READ:RegionName:Key
Region.unregisterInterest(regex)	DATA:READ:RegionName

Client Operation	Assigned `ResourcePermission`
execute function	Defaults to DATA:WRITE. Override `Function.getRequiredPermissions` to change the permission.
clear region	DATA:WRITE:RegionName
Region.putAll	DATA:WRITE:RegionName
Region.clear	DATA:WRITE:RegionName
Region.removeAll	DATA:WRITE:RegionName
Region.destroy(key)	DATA:WRITE:RegionName:Key
Region.invalidate(key)	DATA:WRITE:RegionName:Key
Region.destroy(key)	DATA:WRITE:RegionName:Key
Region.put(key)	DATA:WRITE:RegionName:Key
Region.replace	DATA:WRITE:RegionName:Key
queryService.newCq	DATA:READ:RegionName
CqQuery.stop	CLUSTER:MANAGE:QUERY

This table classifies the permissions assigned for `gfsh` operations.

`gfsh` Command	Assigned `ResourcePermission`
alter async-event-queue	CLUSTER:MANAGE:DEPLOY
alter disk-store	CLUSTER:MANAGE:DISK
alter query-service	CLUSTER:MANAGE
alter region	DATA:MANAGE:RegionName
alter runtime	CLUSTER:MANAGE
backup disk-store	DATA:READ and CLUSTER:WRITE:DISK
change loglevel	CLUSTER:WRITE
clear defined indexes	CLUSTER:MANAGE:QUERY
close durable-client	CLUSTER:MANAGE:QUERY
close durable-cq	CLUSTER:MANAGE:QUERY
compact disk-store	CLUSTER:MANAGE:DISK
configure pdx	CLUSTER:MANAGE
create async-event-queue	CLUSTER:MANAGE:DEPLOY, plus CLUSTER:WRITE:DISK if the associated region is persistent
create defined indexes	CLUSTER:MANAGE:QUERY
create disk-store	CLUSTER:MANAGE:DISK
create gateway-receiver	CLUSTER:MANAGE:GATEWAY

`gfsh` **Command**	**Assigned** `ResourcePermission`
create gateway-sender	CLUSTER:MANAGE:GATEWAY
create index	CLUSTER:MANAGE:QUERY
create jndi-binding	CLUSTER:MANAGE
create lucene index	CLUSTER:MANAGE:LUCENE
create region	DATA:MANAGE, plus CLUSTER:WRITE:DISK if the associated region is persistent
define index	CLUSTER:MANAGE:QUERY
deploy	CLUSTER:MANAGE:DEPLOY
describe client	CLUSTER:READ
describe config	CLUSTER:READ
describe disk-store	CLUSTER:READ
describe jndi-binding	CLUSTER:READ
describe lucene index	CLUSTER:READ:LUCENE
describe member	CLUSTER:READ
describe offline-disk-store	CLUSTER:READ
describe query-service	CLUSTER:READ
describe region	CLUSTER:READ
destroy async-event-queue	CLUSTER:MANAGE
destroy disk-store	CLUSTER:MANAGE:DISK
destroy function	CLUSTER:MANAGE:DEPLOY
destroy index	CLUSTER:MANAGE:QUERY
destroy jndi-binding	CLUSTER:MANAGE
destroy lucene index	CLUSTER:MANAGE:LUCENE
destroy region	DATA:MANAGE
execute function	Defaults to DATA:WRITE. Override `Function.getRequiredPermissions` to change the permission.
export cluster-configuration	CLUSTER:READ
export config	CLUSTER:READ
export data	CLUSTER:READ
export logs	CLUSTER:READ
export offline-disk-store	CLUSTER:READ
export stack-traces	CLUSTER:READ
gc	CLUSTER:MANAGE

gfsh Command	Assigned ResourcePermission
get -key=key1 -region=region1	DATA:READ:RegionName:Key
import data	DATA:WRITE:RegionName
import cluster-configuration	CLUSTER:MANAGE
list async-event-queues	CLUSTER:READ
list clients	CLUSTER:READ
list deployed	CLUSTER:READ
list disk-stores	CLUSTER:READ
list durable-cqs	CLUSTER:READ
list functions	CLUSTER:READ
list gateways	CLUSTER:READ
list indexes	CLUSTER:READ:QUERY
list jndi-binding	CLUSTER:READ
list lucene indexes	CLUSTER:READ:LUCENE
list members	CLUSTER:READ
list regions	CLUSTER:READ
load-balance gateway-sender	CLUSTER:MANAGE:GATEWAY
locate entry	DATA:READ:RegionName:Key
netstat	CLUSTER:READ
pause gateway-sender	CLUSTER:MANAGE:GATEWAY
put --key=key1 --region=region1	DATA:WRITE:RegionName:Key
query	DATA:READ:RegionName
rebalance	DATA:MANAGE
remove	DATA:WRITE:RegionName or DATA:WRITE:RegionName:Key
resume async-event-queue-dispatcher	CLUSTER:MANAGE
resume gateway-sender	CLUSTER:MANAGE:GATEWAY
revoke mising-disk-store	CLUSTER:MANAGE:DISK
search lucene	DATA:READ:RegionName
show dead-locks	CLUSTER:READ
show log	CLUSTER:READ
show metrics	CLUSTER:READ
show missing-disk-stores	CLUSTER:READ

`gfsh` Command	Assigned `ResourcePermission`
show subscription-queue-size	CLUSTER:READ
shutdown	CLUSTER:MANAGE
start gateway-receiver	CLUSTER:MANAGE:GATEWAY
start gateway-sender	CLUSTER:MANAGE:GATEWAY
start server	CLUSTER:MANAGE
status cluster-config-service	CLUSTER:READ
status gateway-receiver	CLUSTER:READ
status gateway-sender	CLUSTER:READ
status locator	CLUSTER:READ
status server	CLUSTER:READ
stop gateway-receiver	CLUSTER:MANAGE:GATEWAY
stop gateway-receiver	CLUSTER:MANAGE:GATEWAY
stop locator	CLUSTER:MANAGE
stop server	CLUSTER:MANAGE
undeploy	CLUSTER:MANAGE:DEPLOY

The `gfsh connect` does not have a permission, as it is the operation that invokes authentication. These `gfsh` commands do not have permission defined, as they do not interact with the cluster:

- `gfsh describe connection`, which describes the `gfsh` end of the connection
- `gfsh debug`, which toggles the mode within `gfsh`
- `gfsh exit`
- `gfsh help`
- `gfsh hint`
- `gfsh history`
- `gfsh run`, although individual commands within the script will go through authorization
- `gfsh set variable`
- `gfsh sh`
- `gfsh sleep`
- `validate offline-disk-store`
- `gfsh version`

This table classifies the permissions assigned for JMX operations.

JMX Operation	Assigned `ResourcePermission`
DistributedSystemMXBean.shutdownAllMembers	CLUSTER:MANAGE

JMX Operation	Assigned `ResourcePermission`
ManagerMXBean.start	CLUSTER:MANAGE
ManagerMXBean.stop	CLUSTER:MANAGE
ManagerMXBean.createManager	CLUSTER:MANAGE
ManagerMXBean.shutDownMember	CLUSTER:MANAGE
Mbeans get attributes	CLUSTER:READ
MemberMXBean.showLog	CLUSTER:READ
DistributedSystemMXBean.changerAlertLevel	CLUSTER:WRITE
ManagerMXBean.setPulseURL	CLUSTER:WRITE
ManagerMXBean.setStatusMessage	CLUSTER:WRITE
CacheServerMXBean.closeAllContinuousQuery	CLUSTER:MANAGE:QUERY
CacheServerMXBean.closeContinuousQuery	CLUSTER:MANAGE:QUERY
CacheServerMXBean.executeContinuousQuery	DATA:READ
CqQuery.execute	DATA:READ:RegionName and CLUSTER:MANAGE:QUERY
CqQuery.executeWithInitialResults	DATA:READ:RegionName and CLUSTER:MANAGE:QUERY
DiskStoreMXBean.flush	CLUSTER:MANAGE:DISK
DiskStoreMXBean.forceCompaction	CLUSTER:MANAGE:DISK
DiskStoreMXBean.forceRoll	CLUSTER:MANAGE:DISK
DiskStoreMXBean.setDiskUsageCriticalPercentage	CLUSTER:MANAGE:DISK
DiskStoreMXBean.setDiskUsageWarningPercentage	CLUSTER:MANAGE:DISK
DistributedSystemMXBean.revokeMissingDiskStores	CLUSTER:MANAGE:DISK
DistributedSystemMXBean.setQueryCollectionsDepth	CLUSTER:MANAGE:QUERY
DistributedSystemMXBean.setQueryResultSetLimit	CLUSTER:MANAGE:QUERY
DistributedSystemMXBean.backupAllMembers	DATA:READ and CLUSTER:WRITE:DISK
DistributedSystemMXBean.queryData	DATA:READ
DistributedSystemMXBean.queryDataForCompressedResult	DATA:READ
GatewayReceiverMXBean.pause	CLUSTER:MANAGE:GATEWAY
GatewayReceiverMXBean.rebalance	CLUSTER:MANAGE:GATEWAY
GatewayReceiverMXBean.resume	CLUSTER:MANAGE:GATEWAY
GatewayReceiverMXBean.start	CLUSTER:MANAGE:GATEWAY
GatewayReceiverMXBean.stop	CLUSTER:MANAGE:GATEWAY

JMX Operation	Assigned `ResourcePermission`
GatewaySenderMXBean.pause	CLUSTER:MANAGE:GATEWAY
GatewaySenderMXBean.rebalance	CLUSTER:MANAGE:GATEWAY
GatewaySenderMXBean.resume	CLUSTER:MANAGE:GATEWAY
GatewaySenderMXBean.start	CLUSTER:MANAGE:GATEWAY
GatewaySenderMXBean.stop	CLUSTER:MANAGE:GATEWAY
LockServiceMXBean.becomeLockGrantor	CLUSTER:MANAGE
MemberMXBean.compactAllDiskStores	CLUSTER:MANAGE:DISK

Implement Authorization

Complete these items to implement authorization.

- Decide upon an authorization algorithm. The Authorization Example stores a mapping of which principals (users) are permitted to do which operations. The algorithm bases its decision on a look up of the permissions granted to the principal attempting the operation.
- Define the `security-manager` property. See Enable Security with Property Definitions for details about this property.
- Implement the `authorize` method of the `SecurityManager` interface.
- Define any extra resources that the implemented authorization algorithm needs in order to make a decision.

Authorization of Function Execution

By default, a function executed on servers requires that the entity invoking the function have `DATA:WRITE` permission on the region(s) involved. Since the default permission may not be appropriate for all functions, the permissions required may be altered.

To implement a different set of permissions, override the `Function.getRequiredPermissions()` method in the function's class. The method should return a `Collection` of the permissions required of the entity that invokes an execution of the function.

Authorization of Methods Invoked from Queries

Enabling the `SecurityManager` affects queries by restricting the methods that a running query may invoke. See Method Invocations and Method Invocation Authorizers for details.

Method Invocation Authorizers

Overview

When the `SecurityManager` is enabled, by default VMware GemFire throws a `NotAuthorizedException` when a `method` within a query is invoked and does not belong to the list of default allowed methods, given in RestrictedMethodAuthorizer.

The `MethodInvocationAuthorizer` is used to determine whether a specific method invocation on a given object should be allowed or denied during the execution of a particular OQL query.

Allowing users to execute arbitrary methods on any object present within the VMware GemFire member's classpath could impact the integrity of the data and the system on which VMware GemFire is running. In order to avoid this problem, it is always recommended to enable a `SecurityManager` at the cluster level, give users only the permissions they require, and configure a `MethodInvocationAuthorizer` that meets your needs.

The main threats to which a VMware GemFire cluster might be exposed without a `MethodInvocationAuthorizer` are highlighted below.

Java Reflection

Allows the user to do anything within the JVM.

```
SELECT * FROM /region r WHERE r.getClass().forName('java.lang.Runtime').getDeclaredMet
hods()[0].invoke()
```

Cache Modification

Allows the user to do anything with the `Cache`: close it, access other regions, etc.

```
SELECT * FROM /region.getCache().close()
```

Region Modification

Allows the user to destroy, add or invalidate the entire `Region`, or specific `entries`.

```
SELECT * FROM /region.destroyRegion()
SELECT * FROM /region.put('xyz','abc')
SELECT * FROM /region.invalidate('xyz')
```

Region Modification

Allows the user to mutate the state of specific `entries`.

```
`SELECT r.setName('newName') FROM /region r`
|
```

VMware GemFire Authorizers

VMware GemFire provides four authorizers out of the box, each one designed and implemented for a specific use case in mind. It is recommended to always use one of these authorizers, and only implement your own if your use case needs are not already met by one of them.

All of the implementations provided by VMware GemFire are designed to prevent security problems and have been thoroughly tested. Extra care should be taken, however, when configuring the internals of some of the authorizers as an incorrect configuration might introduce security holes into the system.

The table below shows a summary of which security threats are fully addressed by each authorizer and which ones might be exploitable, depending on how they are configured (details are shown later for each implementation).

Authorizer \ Threat	Reflection	Cache Access	Region Access	Entry Modification
RestrictedMethodAuthorizer	✓	✓	✓	✓
UnrestrictedMethodAuthorizer	✓	✓	✓	✗
RegExMethodAuthorizer	✓	✓	✓	✗
JavaBeanAccessorMethodAuthorizer	✓	✓	✓	✗

RestrictedMethodAuthorizer

The default `MethodInvocationAuthorizer` used by VMware GemFire to determine whether a method is allowed to be executed on a specific object instance or not.

The implementation forbids the invocation of all methods during a query execution, except for the ones shown below:

Class	Allowed Methods
`java.lang.Object`	`equals`, `toString`, `compareTo`
`java.lang.Boolean`	`booleanValue`
`java.lang.Number`	`byteValue`, `intValue`, `doubleValue`, `floatValue`, `longValue`, `shortValue`
`java.util.Date`	`after`, `before`, `getTime`
`java.sql.Timestamp`	`getNanos`
`java.lang.String`	`chartAt`, `codePointAt`, `codePointBefore`, `codePointCount`, `compareToIgnoreCase`, `concat`, `contains`, `contentEquals`, `endsWith`, `equalsIgnoreCase`, `getBytes`, `hashCode`, `indexOf`, `intern`, `isEmpty`, `lastIndexOf`, `length`, `matches`, `offsetByCodePoints`, `replace`, `replaceAll`, `replaceFirst`, `split`, `startsWith`, `substring`, `toCharArray`, `toLowerCase`, `toUpperCase`, `trim`
`java.util.Map.Entry`, `org.apache.geode.cache.Region.Entry`	`getKey`, `getValue`
`java.util.Collection`, `java.util.Map`, `org.apache.geode.cache.Region`	`get`, `entrySet`, `keySet`, `values`, `getEntries`, `getValues`, `containsKey`

The authorizer also provides utilities that can be used by custom implementations to determine whether a method is permanently forbidden or, if the method belongs to VMware GemFire, whether it is considered safe to be used within a query execution.

The methods `getClass`, `readObject`, `readResolve`, `readObjectNoData`, `writeObject` and `writeReplace` are permanently forbidden.

The below table shows those methods that belong to VMware GemFire and are considered safe (for methods on `org.apache.geode.cache.Region`, the authorizer also verifies that the user has the `DATA:READ:RegionName` permission).

Class	Allowed Methods
`org.apache.geode.cache.Region.Entry`	`getKey`, `getValue`

Class	Allowed Methods
`org.apache.geode.cache.Region`	`get`, `entrySet`, `keySet`, `values`, `getEntries`, `getValues`, `containsKey`

UnrestrictedMethodAuthorizer

A less restrictive `MethodInvocationAuthorizer` that allows any method invocation during the query execution as long as the following conditions are met:

- The method is not considered permanently forbidden by the RestrictedMethodAuthorizer.
- The method does not belong to VMware GemFire, or does belong but is considered safe by the RestrictedMethodAuthorizer.

This authorizer implementation addresses only three of the four main security risks: `Java Reflection`, `Cache Modification` and `Region Modification`. The `Region Entry Modification` security risk still exists: users with the `DATA:READ:RegionName` permission will be able to execute ANY method (even those that mutate the object) on the entries stored within the region and on instances used as bind parameters of the query, so this authorizer implementation must be used with extreme care.

Note: Usage of this authorizer is recommended for secured clusters on which only trusted users and applications have access to the query engine. It might also be used on clusters on which all entries stored are immutable.

JavaBeanAccessorMethodAuthorizer

A more flexible `MethodInvocationAuthorizer` that allows methods to be invoked during a query execution if and only if all of the following conditions are met:

- The method is not considered permanently forbidden by the RestrictedMethodAuthorizer.
- The method does not belong to VMware GemFire, or does belong but is considered safe by the RestrictedMethodAuthorizer.
- The method follows the design patterns for accessor methods described in the JavaBean Specification 1.01; that is, the method name begins with `is` or `get`.
- The target object on which the method will be executed belongs to a set of pre-configured packages.

When used as intended, and assuming that all region entries and bind parameters follow the JavaBean Specification 1.01, this authorizer implementation addresses all four security risks: `Java Reflection`, `Cache Modification`, `Region Modification` and `Region Entry Modification`. It should be noted, however, that the `Region Entry Modification` security threat might be re-introduced: users with the `DATA:READ:RegionName` privilege will be able to execute any method whose name starts with `is` or `get` on the objects stored within the region and on instances used as bind parameters, providing they are in the pre-configured packages. If those methods do not fully follow the JavaBean Specification 1.01 in that **accessors do not mutate the object state**, then instances could be potentially modified in place.

Note: Usage of this authorizer is only recommended for secured clusters on which the user has full confidence in that all objects stored within the regions and used as bind parameters follow the

JavaBean Specification 1.01. It might also be used on clusters on which all entries stored are immutable.

RegExMethodAuthorizer

A fully flexible `MethodInvocationAuthorizer` that allows methods to be invoked during the query execution only if the the following conditions are met:

- The method is not considered permanently forbidden by the RestrictedMethodAuthorizer.
- The method does not belong to VMware GemFire, or does belong but is considered safe by the RestrictedMethodAuthorizer.
- The fully qualified method name matches at least one of the pre-configured regular expressions.

When correctly configured, this authorizer implementation addresses the four main security risks: `Java Reflection`, `Cache Modification`, `Region Modification` and `Region Entry Modification`. For the statement to remain true, however, the regular expressions used must be correctly configured so no mutator methods ever match. If the regular expressions are not restrictive enough, the `Region Entry Modification` security risk might be potentially re-introduced: users with the `DATA:READ:RegionName` privilege will be able to execute methods (even those modifying the entry) on the objects stored within the region and on instances used as bind parameters of the query.

Note: This authorizer must be used with extreme care, it is the most powerful in terms of flexibility and versatility (full control through regular expressions regarding what to allow and what to forbid); but it is also the most dangerous as one small mistake in the configured regular expressions can unexpectedly allow a wide variety of non safe methods to be executed.

Note: Usage of this authorizer implementation is only recommended for scenarios in which the user knows exactly what code is deployed to the cluster, allowing a correct configuration of the regular expressions used. It might also be used on clusters on which all entries stored are immutable.

Custom Authorizers

How Authorization Works

It is important to note that the query engine does not have any information about the actual type of the objects while pre-processing or parsing the query itself, neither can it obtain these details before actually executing the query. The actual check to determine whether a method is allowed or not must be executed while the objects are being traversed by the query engine in runtime.

The query engine, however, remembers whether a specific method has been already authorized or not for the current query execution context, meaning that **the authorization will be executed only once in the lifetime of a particular query for every new method seen while traversing the objects**. Nevertheless, the authorizer implementation must be highly performant as it will be invoked by VMware GemFire in runtime during the actual query execution.

Implementing a Method Authorizer

Complete these items to implement a custom method authorizer.

- Decide which methods from your domain model should be allowed to be invoked during a query execution.
- Decide which resources, if any, you will need in order to determine whether a method can be invoked or not.
- Implement the `initialize` method of the `MethodInvocationAuthorizer` interface to fully configure your implementation, based on the resources needed to execute the authorization.
- Implement the `authorize` method of the `MethodInvocationAuthorizer` interface. It must determine whether a `method` is allowed to be executed on a particular object instance during a query execution. The implementation should be **lightning fast** and **thread safe**.

Changing the Method Authorizer

You can set the `MethodInvocationAuthorizer` to be used by the query engine through the `gfsh` command-line utility. In addition, you can modify the configured `MethodInvocationAuthorizer` while members are already running by using the alter query-service command. It is always advisable to make these changes during periods of low activity, though.

The following constraints apply when the `MethodInvocationAuthorizer` used by the cluster is changed in runtime:

- Queries started after the `MethodInvocationAuthorizer` is changed will use the newly configured authorizer.
- Queries in flight are not affected. Before the query starts, it picks up the already configured `MethodInvocationAuthorizer` and will use it until the execution finishes.
- Indexes configured with an expression using methods forbidden by the newly configured `MethodInvocationAuthorizer` will be marked as invalid the next time a mapping is added or removed from the index.
- Continuous queries already running will pick up the newly configured `MethodInvocationAuthorizer` the next time the CQ is processed upon the arrival of a cache event. If the CQ has methods forbidden by the newly configured `MethodInvocationAuthorizer`, any subsequent execution will result in an error during the CQ processing, and the `onError` method will be invoked for the associated `CqListener`.

Note: In order to improve performance, the continuous query engine uses an internal cache to avoid executing the query in scenarios for which the answer can be automatically inferred. These results might become invalid after applying the new security rules, so VMware GemFire deactivates the usage of this optimization until the member is restarted or the query is registered again.

Authorization Example

Disclaimer

The security implementation of every installation is unique. These examples are provided for illustrative purposes only and must not be used in a production environment.

The examples demonstrate the basics for implementing both user authorization (`SecurityManager.authorize`) and method invocation authorization (`MethodInvocationAuthorizer.authorize`) during query executions. The remainder of the examples may be found within the VMware GemFire source code under the `geode-core/src/main/java/org/apache/geode/examples/security` directory.

User Authorization Example

This example assumes that a set of users, a set of roles that a user might take on within the system, and a mapping of users to their roles are described in a JSON format file. The roles define a set of authorized resource permissions granted for users in those roles. Code not shown here parses the file to compose a data structure with the information on roles and users. The `authorize` callback denies permission for any operation that does not have a principal representing the identity of the operation's requester. Given the principal, the method iterates through the data structure searching for the necessary permissions for the principal. When the necessary permission is found, authorization is granted by returning the value `true`. If the permission is not found in the data structure, then the method returns `false`, denying authorization of the operation.

```
public boolean authorize(final Object principal, final ResourcePermission context) {
    if (principal == null) return false;

    User user = this.userNameToUser.get(principal.toString());
    if (user == null) return false; // this user is not authorized to do anything

    // check if the user has this permission defined in the context
    for (Role role : this.userNameToUser.get(user.name).roles) {
        for (Permission permitted : role.permissions) {
            if (permitted.implies(context)) {
                return true;
            }
        }
    }

    return false;
}
```

Method Invocation Authorization Example

This example assumes that the entire domain model is deployed to the cluster and that the user is allowed to modify these classes. The `authorize` callback denies access to methods that have been permanently forbidden by the RestrictedMethodAuthorizer and returns `false` right away. When the method is not permanently forbidden, the implementation checks whether the method has been annotated with a custom annotation. When the necessary annotation is found, authorization is granted by returning the value `true`. If the annotation is not found, then the method returns `false`, denying the invocation of the method during the query execution.

```
public boolean authorize(Method method, Object target) {
    // Check if forbidden by default.
    if (defaultAuthorizer.isPermanentlyForbiddenMethod(method, target)) {
        return false;
    }
    // Check if annotation is present
    return method.isAnnotationPresent(Authorized.class);
  }
```

Post Processing of Region Data

The `PostProcessor` interface allows the definition of a callback that is invoked after any and all client and `gfsh` operations that get data, but before the data is returned. It permits the callback to intervene and format the data that is to be returned. The callbacks must not modify the region data, only the data to be returned.

The `processRegionValue` method is given the principal of the operation requester. The operation will already have been completed, implying that the principal will have been authorized to complete the requested operation. The post processing can therefore format the returned data based on the identity of the requester (principal).

By default, the key and value parameters to the `processRegionValue` method are references to the region entry. Modify copies of these parameters to avoid changing the region entries. Copy on Read Behavior discusses the issue.

The `processRegionValue` method is invoked for these API calls:

- `Region.get`
- `Region.getAll`
- `Query.execute`
- `CqQuery.execute`
- `CqQuery.executeWithInitialResults`
- `CqListener.onEvent`
- for a relevant region event from `CacheListener.afterUpdate` for which there is interest registered with `Region.registerInterest`

Care should be taken when designing a system that implements the post processing callback. It incurs the performance penalty of an extra method invocation on every get operation.

Implement Post Processing

Complete these items to implement post processing.

- Define the `security-post-processor` property. See Enable Security with Property Definitions for details about this property.
- Implement the `processRegionValue` method of the `PostProcessor` interface.

SSL

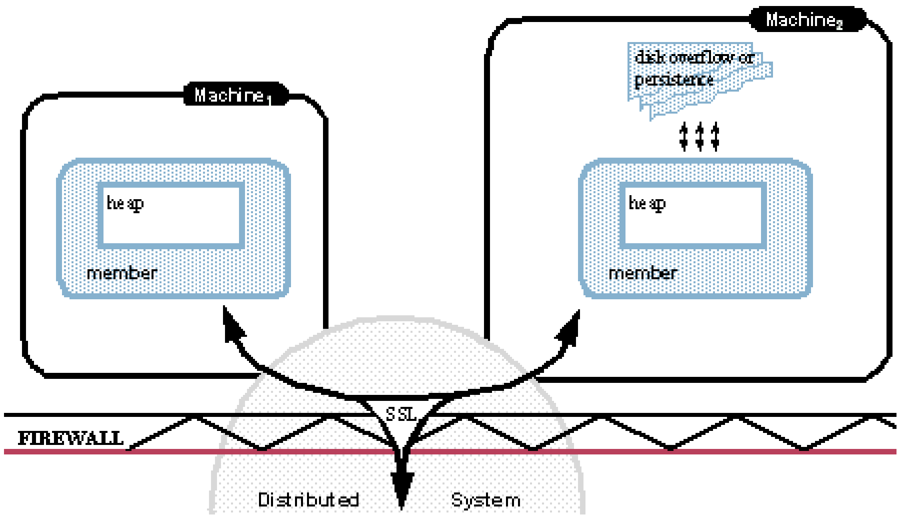
SSL protects your data in transit between applications by ensuring that only the applications identified by you can share cluster data.

To be secure, the data that is cached in a VMware GemFire system must be protected during storage, distribution, and processing. At any time, data in a cluster may be in one or more of these locations:

- In memory
- On disk
- In transit between processes (for example, in an internet or intranet)

For the protection of data in memory or on disk, VMware GemFire relies on your standard system security features such as firewalls, operating system settings, and JDK security settings.

The SSL implementation ensures that only the applications identified by you can share cluster data in transit. In this figure, the data in the visible portion of the cluster is secured by the firewall and by security settings in the operating system and in the JDK. The data in the disk files, for example, is protected by the firewall and by file permissions. Using SSL for data distribution provides secure communication between VMware GemFire system members inside and outside the firewalls.



- **Configuring SSL**

  You configure SSL for mutual authentication between members and to protect your data during distribution. You can use SSL alone or in conjunction with the other VMware GemFire security options.

- **SSL Sample Implementation**

  A simple example demonstrates the configuration and startup of VMware GemFire system components with SSL.

Configuring SSL

You can configure SSL for authentication between members and to protect your data during distribution. You can use SSL alone or in conjunction with the other VMware GemFire security options.

VMware GemFire SSL connections use the Java Secure Sockets Extension (JSSE) package, so the properties described here apply to VMware GemFire servers and to Java-based clients. SSL configuration in non-Java clients may differ — see the client’s documentation for details.

SSL-Configurable Components

You can specify that SSL be used system-wide, or you can independently configure SSL for specific system components. The following list shows the system components that can be separately configured to communicate using SSL, and the kind of communications to which each component name refers:

cluster
: Peer-to-peer communications among members of a cluster

gateway
: Communication across WAN gateways from one site to another

web
: All web-based services hosted on the configured server, which can include the Developer REST API service, the Management REST API service (used for remote cluster management) and the Pulse monitoring tool's web-based user interface.

jmx
: Java management extension communications, including communications with the `gfsh` utility. The Pulse monitoring tool uses JMX for server-side communication with a locator, but SSL applies to this connection only if Pulse is located on an app server separate from the locator. When Pulse and the locator are colocated, JMX communication between the two does not involve a TCP connection, so SSL does not apply.

locator
: Communication with and between locators

server
: Communication between clients and servers

all
: All of the above (use SSL system-wide)

Specifying that a component is enabled for SSL applies to the component’s server-socket side and its client-socket side. For example, if you enable SSL for locators, then any process that communicates with a locator must also have SSL enabled. If you provide "" as the value, SSL is turned off for all components.

SSL Configuration Properties

You can use VMware GemFire configuration properties to enable or disable SSL, to identify SSL ciphers and protocols, and to provide the location and credentials for key and trust stores.

ssl-enabled-components
: List of components for which to enable SSL. Component list can be "" (disable SSL), "all", or a comma-separated list of components.

ssl-endpoint-identification-enabled
: A boolean value that, when set to true, causes clients to validate the server's hostname using the server's certificate. The default value is false. Enabling endpoint identification guards against DNS man-in-the-middle attacks when trusting certificates that are not self-signed.

ssl-use-default-context
: A boolean value that, when set to true, allows VMware GemFire to use the default SSL context as returned by SSLContext.getInstance('Default') or set by using SSLContext.setDefault(). When enabled, also causes ssl-endpoint-identification-enabled to be set to true.

ssl-require-authentication
: Requires two-way authentication, applies to all components except web. Boolean - if true (the default), two-way authentication is required.

ssl-web-require-authentication
: Requires two-way authentication for web component. Boolean - if true, two-way authentication is required. Default is false (one-way authentication only).

ssl-default-alias
: A server uses one key store to hold its SSL certificates. All components on that server can share a single certificate, designated by the ssl-default-alias property. If ssl-default-alias is not specified, the first certificate in the key store acts as the default certificate.

ssl-_component_-alias=string
: You can configure a separate certificate for any component. All certificates reside in the same key store, but can be designated by separate aliases that incorporate the component name, using this syntax, where _component_ is the name of a component. When a component-specific alias is specified, it overrides the ssl-default-alias for the _component_ specified. For example, ssl-locator-alias would specify a name for the locator component's certificate in the system key store.

ssl-ciphers
: A comma-separated list of the valid ciphers for TCP/IP connections with TLS encryption enabled. A setting of 'any' allows the JSSE provider to select an appropriate cipher that it supports.

ssl-protocols
: A comma-separated list of the valid protocol versions for TCP/IP connections with TLS encryption enabled. A setting of 'any' attempts to use your JSSE provider's TLSv1.3, or TLSv1.2 if v1.3 is not available.

ssl-keystore, ssl-keystore-password
: The path to the key store and the key store password, specified as strings

ssl-truststore, ssl-truststore-password
: The path to the trust store and the trust store password, specified as strings

ssl-keystore-type, ssl-truststore-type
: The types of the key store and trust store, specified as strings. The default for both is "JKS", indicating a Java key store or trust store.

Example: secure communications throughout

To implement secure SSL communications throughout an entire cluster, each process should enable SSL for all components.

```
ssl-enabled-components=all
ssl-endpoint-identification-enabled=true
ssl-keystore=secure/keystore.dat
ssl-keystore-password=changeit
ssl-truststore=secure/truststore.dat
ssl-truststore-password=changeit
```

If the key store has multiple certificates you may want to specify the alias of the one you wish to use for each process. For instance, `ssl-default-alias=Hiroki`.

Example: non-secure cluster communications, secure client/server

In this example, SSL is used to secure communications between the client and the server:

Server properties

Cluster SSL is not enabled.

```
ssl-enabled-components=server,locator
ssl-server-alias=server
ssl-keystore=secure/keystore.dat
ssl-keystore-password=changeit
ssl-truststore=secure/truststore.dat
ssl-truststore-password=changeit
ssl-default-alias=Server-Cert
```

Locator properties

Cluster SSL is not enabled.

```
ssl-enabled-components=locator
ssl-locator-alias=locator
ssl-keystore=secure/keystore.dat
ssl-keystore-password=changeit
ssl-truststore=secure/truststore.dat
ssl-truststore-password=changeit
ssl-default-alias=Locator-Cert
```

Client properties

On Java clients, the list of enabled components reflects the server's configuration so the client knows how it is expected to communicate with (for example) servers and locators. Paths to keystore and truststore are local to the client.

In this example, the client's trust store must trust both locator and server certificates. Since the client does not specify a certificate alias, SSL will use the default certificate in its key store.

```
ssl-enabled-components=server,locator
ssl-endpoint-identification-enabled=true
ssl-keystore=secret/keystore.dat
ssl-keystore-password=changeit
ssl-truststore=secret/truststore.dat
ssl-truststore-password=changeit
```

SSL Property Reference Tables

The following table lists the components you can configure to use SSL.

Table 1. SSL-Configurable Components

Component	Communication Types
cluster	Peer-to-peer communications among members of a cluster
gateway	Communication across WAN gateways from one site to another
web	Web-based communication, including REST interfaces
jmx	Java management extension communications, including gfsh

Component	Communication Types
locator	Communication with and between locators
server	Communication between clients and servers
all	All of the above

The following table lists the properties you can use to configure SSL on your VMware GemFire system.

Table 2. SSL Configuration Properties

Property	Description	Value
ssl-enabled-components	list of components for which to enable SSL	“all”, "", or comma-separated list of components: cluster, gateway, web, jmx, locator, server
ssl-endpoint-identification-enabled	causes clients to validate server hostname using server certificate	boolean - if true, does validation; defaults to false
ssl-use-default-context	allows VMware GemFire to use the default SSL context	boolean - if true, uses the default SSL context. Also sets ssl-endpoint-identification-enabled to true; defaults to false
ssl-require-authentication	requires two-way authentication, applies to all components except web	boolean - if true (the default), two-way authentication is required
ssl-web-require-authentication	requires two-way authentication for web component	boolean - if true, two-way authentication is required. Default is false (one-way authentication only)
ssl-default-alias	default certificate name	string - if empty, use first certificate in key store
ssl-*component*-alias	component-specific certificate name	string - applies to specified *component*
ssl-ciphers	list of SSL ciphers	comma-separated list (default “any”)
ssl-protocols	list of SSL protocols	comma-separated list (default “any”)
ssl-keystore	path to key store	string
ssl-keystore-password	key store password	string
ssl-keystore-type	trust store type	string
ssl-truststore	path to trust store	string
ssl-truststore-password	trust store password	string
ssl-truststore-type	trust store type	string

Procedure

1. Make sure your Java installation includes the JSSE API and familiarize yourself with its use. For information, see the Oracle JSSE website.
2. Configure SSL as needed for each connection type:

1. Use locators for member discovery within the clusters and for client discovery of servers. See Configuring Peer-to-Peer Discovery and Configuring a Client/Server System.
2. Configure SSL properties as necessary for different component types, using the properties described above. For example, to enable SSL for communication between clients and servers you would configure properties in the `gemfire.properties` file similar to:

   ```
   ssl-enabled-components=server
   ssl-protocols=any
   ssl-ciphers=SSL_RSA_WITH_NULL_MD5, SSL_RSA_WITH_NULL_SHA
   ssl-keystore=/path/to/trusted.keystore
   ssl-keystore-password=password
   ssl-truststore=/path/to/trusted.keystore
   ssl-truststore-password=password
   ```

SSL Sample Implementation

A simple example demonstrates the configuration and startup of VMware GemFire system components with SSL.

Provider-Specific Configuration File

This example uses a keystore created by the Java `keytool` application to provide the proper credentials to the provider. To create the keystore, run the `keytool` utility:

```
keytool -genkey \
-alias self \
-dname "CN=trusted" \
-validity 3650 \
-keypass password \
-keystore ./trusted.keystore \
-storepass password \
-storetype JKS
```

This creates a `./trusted.keystore` file to be used later.

gemfire.properties File

You can enable SSL in the `gemfire.properties` file. In this example, SSL is enabled for all components.

```
ssl-enabled-components=all
mcast-port=0
locators=<hostaddress>[<port>]
```

gfsecurity.properties File

You can specify the provider-specific settings in a `gfsecurity.properties` file, which can then be secured by restricting access to this file. The following example configures the default JSSE provider settings included with the JDK.

```
ssl-keystore=/path/to/trusted.keystore
ssl-keystore-password=password
ssl-truststore=/path/to/trusted.keystore
ssl-truststore-password=password
security-username=xxxx
security-userPassword=yyyy
```

Locator Startup

Before starting other system members, we started the locator with the SSL and provider-specific configuration settings. After properly configuring `gemfire.properties` and `gfsecurity.properties`, start the locator and provide the location of the properties files. If any of the password fields are left empty, you will be prompted to enter a password.

```
gfsh>start locator --name=my_locator --port=12345 \
--properties-file=/path/to/your/gemfire.properties \
--security-properties-file=/path/to/your/gfsecurity.properties
```

Other Member Startup

Applications and cache servers can be started similarly to the locator startup, with the appropriate `gemfire.properties` file and `gfsecurity.properties` files placed in the current working directory. You can also pass in the location of both files as system properties on the command line. For example:

```
gfsh>start server --name=my_server \
--properties-file=/path/to/your/gemfire.properties \
--security-properties-file=/path/to/your/gfsecurity.properties
```

Connecting to a Running Cluster

You can use `gfsh` to connect to an SSL-enabled cluster that is already running by specifying the `use-ssl` command-line option and providing a path to the security configuration file:

```
gfsh>connect --locator=localhost[10334] --use-ssl \
--security-properties-file=/path/to/your/gfsecurity.properties
```

Once connected, you can then issue `gfsh` commands to perform a variety of operations, including listing members and displaying region characteristics.

Performance Tuning and Configuration

A collection of tools and controls allow you to monitor and adjust VMware GemFire performance.

- **Improving Performance on vSphere**

  This topic provides guidelines for tuning vSphere virtualized environments that host VMware GemFire deployments.

- **Performance Controls**

  This topic provides tuning suggestions of particular interest to developers, primarily programming techniques and cache configuration.

- **System Member Performance**

  You can modify some configuration parameters to improve system member performance.

- **Slow Receivers with TCP/IP**

  You have several options for preventing situations that can cause slow receivers of data distributions. The slow receiver options control only peer-to-peer communication using TCP/IP. This discussion does not apply to client/server or multi-site communication, or to communication using the UDP unicast or multicast protocols.

- **Slow distributed-ack Messages**

  In systems with distributed-ack regions, a sudden large number of distributed-no-ack operations can cause distributed-ack operations to take a long time to complete.

- **Socket Communication**

  VMware GemFire processes communicate using TCP/IP and UDP unicast and multicast protocols. In all cases, communication uses sockets that you can tune to optimize performance.

- **UDP Communication**

  You can make configuration adjustments to improve multicast and unicast UDP performance of peer-to-peer communication.

- **Multicast Communication**

  You can make configuration adjustments to improve the UDP multicast performance of peer-to-peer communication in your VMware GemFire system.

- **Maintaining Cache Consistency**

  Maintaining data consistency between caches in a distributed VMware GemFire system is vital for ensuring its functional integrity and preventing data loss.

Improving Performance on vSphere

Operating System Guidelines

Use the latest supported version of the guest OS, and use Java large paging.

- **Use the latest supported version of the guest operating system**. This guideline is probably the most important. Upgrade the guest OS to a recent version supported by VMware GemFire. For example, for RHEL, use at least version 7.0 or for SLES, use at least 11.0. For Windows, use Windows Server 2012. For RedHat Linux users, it is particularly beneficial to use RHEL 7 since there are specific enhancements in the RHEL 7 release that improve virtualized latency sensitive workloads.
- **Use Java large paging in guest OS**. Configure Java on the guest OS to use large pages. Add the following command line option when launching Java:

```
-XX:+UseLargePages
```

NUMA, CPU, and BIOS Settings

This section provides VMware-recommended NUMA, CPU, and BIOS settings for your hardware and virtual machines.

- Always enable hyper-threading, and do not overcommit CPU.
- For most production VMware GemFire servers, always use virtual machines with at least two vCPUs .
- Apply non-uniform memory access (NUMA) locality by sizing virtual machines to fit within the NUMA node.
- VMware recommends the following BIOS settings:
  - **BIOS Power Management Mode:** Maximum Performance.
  - **CPU Power and Performance Management Mode:** Maximum Performance.
  - **Processor Settings:**Turbo Mode enabled.
  - **Processor Settings:**C States disabled.

Note: Settings may vary slightly depending on your hardware make and model. Use the settings above or equivalents as needed.

Physical and Virtual NIC Settings

These guidelines help you reduce latency.

- **Physical NIC:** VMware recommends that you disable interrupt coalescing on the physical NIC of your ESXi host by using the following command:

  ```
  ethtool -C vmnicX rx-usecs 0 rx-frames 1 rx-usecs-irq 0 rx-frames-irq 0
  ```

  where `vmnicX` is the physical NIC as reported by the ESXi command:

  ```
  esxcli network nic list
  ```

  You can verify that your settings have taken effect by issuing the command:

  ```
  ethtool -C vmnicX
  ```

  If you restart the ESXi host, the above configuration must be reapplied.

  Note: Disabling interrupt coalescing can reduce latency in virtual machines; however, it can impact performance and cause higher CPU utilization. It can also defeat the benefits of large receive offloads (LRO) because some physical NICs (such as Intel 10GbE NICs) automatically disable LRO when interrupt coalescing is disabled. This type of tuning benefits VMware GemFire workloads, but it can hurt other non-VMware GemFire workloads that are memory throughput-bound, as opposed to latency sensitive as in the case of VMware GemFire workloads. See http://kb.vmware.com/kb/1027511 for more details.

- **Virtual NIC:** Use the following guidelines when configuring your virtual NICs:
    - Use VMXNET3 virtual NICs for your latency-sensitive or otherwise performance-critical virtual machines. See http://kb.vmware.com/kb/1001805 for details on selecting the appropriate type of virtual NIC for your virtual machine.
    - VMXNET3 supports adaptive interrupt coalescing that can help drive high throughput to virtual machines that have multiple vCPUs with parallelized workloads (multiple threads), while minimizing latency of virtual interrupt delivery. However, if your workload is extremely sensitive to latency, VMware recommends that you disable virtual interrupt coalescing for your virtual NICs. You can do this programmatically via API or by editing your virtual machine's .vmx configuration file. Refer to your vSphere API Reference or VMware ESXi documentation for specific instructions.

VMware vSphere vMotion and DRS Cluster Usage

This topic discusses use limitations of vSphere vMotion, including its use with DRS.

When vMotion migrations occur, there is an expected temporary drop in the performance of both read-operation and write-operation workloads. These workloads resume their normal rate of operation once the vMotion migration of the servers is completed.

VMware recommends that all vMotion migration activity of VMware GemFire members occurs over 10GbE, during periods of low activity and scheduled maintenance windows. Test vMotion migrations in your own environment to assess differences in workload, networking, and scale.

If you wish to prevent automatic VMware vSphere vMotion® operations that can affect response times, place VMware vSphere Distributed Resource Scheduler™ (DRS) in manual mode when you first commission the data management system.

Placement and Organization of Virtual Machines

This section provides guidelines on JVM instances and placement of redundant copies of cached data.

- Have one JVM instance per virtual machine.
- Increasing the heap space to service the demand for more data is better than installing a second instance of a JVM on a single virtual machine. If increasing the JVM heap size is not an option, consider placing the second JVM on a separate newly created virtual machine, thus promoting more effective horizontal scalability. As you increase the number of VMware GemFire servers, also increase the number of virtual machines to maintain a 1:1:1 ratio among the VMware GemFire server, the JVM, and the virtual machines.
- Size for a minimum of four vCPU virtual machines with one VMware GemFire server running in one JVM instance. This allows ample CPU cycles for the garbage collector, and the rest for user transactions.
- Because VMware GemFire can place redundant copies of cached data on any virtual machine, it is possible to inadvertently place two redundant data copies on the same ESX/ESXi host. This is not optimal if a host fails. To create a more robust configuration, use

VM1-to-VM2 anti-affinity rules, to indicate to vSphere that VM1 and VM2 can never be placed on the same host because they hold redundant data copies.

Virtual Machine Memory Reservation

This section provides guidelines for sizing and setting memory.

- Set memory reservation at the virtual machine level so that ESXi provides and locks down the needed physical memory upon virtual machine startup. Once allocated, ESXi does not allow the memory to be taken away.
- Do not overcommit memory for VMware GemFire hosts.
- When sizing memory for a VMware GemFire server within one JVM on one virtual machine, the total reserved memory for the virtual machine should not exceed what is available within one NUMA node for optimal performance.

vSphere High Availability and VMware GemFire

On VMware GemFire virtual machines, disable vSphere High Availability (HA).

If you are using a dedicated VMware GemFire DRS cluster, then you can disable HA across the cluster. However, if you are using a shared cluster, exclude VMware GemFire virtual machines from vSphere HA.

Additionally, to support high availability, you can also set up anti-affinity rules between the VMware GemFire virtual machines to prevent two VMware GemFire servers from running on the same ESXi host within the same DRS cluster.

Storage Guidelines

This section provides storage guidelines for persistence files, binaries, logs, and more.

- Use the PVSCSI driver for I/O intensive VMware GemFire workloads.
- Align disk partitions at the VMFS and guest operating system levels.
- Provision VMDK files as eagerzeroedthick to avoid lazy zeroing for VMware GemFire members.
- Use separate VMDKs for VMware GemFire persistence files, binaries, and logs.
- Map a dedicated LUN to each VMDK.
- For Linux virtual machines, use NOOP scheduling as the I/O scheduler instead of Completely Fair Queuing (CFQ). Starting with the Linux kernel 2.6, CFQ is the default I/O scheduler in many Linux distributions. See http://kb.vmware.com/kb/2011861 for more information.

Additional Resources

These older VMware publications provide additional resources on optimizing for vSphere.

- “Performance Best Practices for VMware vSphere 5.0” - http://www.vmware.com/pdf/Perf_Best_Practices_vSphere5.0.pdf

- “Best Practices for Performance Tuning of Latency-Sensitive Workloads in vSphere Virtual Machines” - http://www.vmware.com/files/pdf/techpaper/VMW-Tuning-Latency-Sensitive-Workloads.pdf
- “Enterprise Java Applications on VMware - Best Practices Guide” - http://www.vmware.com/resources/techresources/1087

Performance Controls

This topic provides tuning suggestions of particular interest to developers, primarily programming techniques and cache configuration.

Before you begin, you should understand VMware GemFire Basic Configuration and Programming.

- **Data Serialization**

  In addition to standard Java serialization, VMware GemFire offers serialization options that give you higher performance and greater flexibility for data storage, transfers, and language types.

- **Setting Cache Timeouts**

  Cache timeout properties can modified through the gfsh `alter runtime` command (or declared in the `cache.xml` file) and can also be set through methods of the interface, `org.apache.geode.cache.Cache`.

- **Controlling Socket Use**

  For peer-to-peer communication, you can manage socket use at the system member level and at the thread level.

- **Management of Slow Receivers**

  You have several options for handling slow members that receive data distribution. The slow receiver options control only to peer-to-peer communication between distributed regions using TCP/IP. This topic does not apply to client/server or multi-site communication, or to communication using the UDP unicast or IP multicast protocols.

- **Increasing the Ratio of Cache Hits**

  The more frequently a get fails to find a valid value in the first cache and has to try a second cache, the more the overall performance is affected.

Data Serialization

In addition to standard Java serialization, VMware GemFire offers serialization options that give you higher performance and greater flexibility for data storage, transfers, and language types.

Under *Developing with VMware GemFire*, see Data Serialization.

Setting Cache Timeouts

Cache timeout properties can modified through the gfsh `alter runtime` command (or declared in the `cache.xml` file) and can also be set through methods of the interface, `org.apache.geode.cache.Cache`.

To modify cache timeout properties, you can issue the following `gfsh alter runtime` command. For example:

```
gfsh>alter runtime --search-timeout=150
```

The `--search-timeout` parameter specifies how long a netSearch operation can wait for data before timing out. The default is 5 minutes. You may want to change this based on your knowledge of the network load or other factors.

The next two configurations describe timeout settings for locking in regions with global scope. Locking operations can time out in two places: when waiting to obtain a lock (lock time out); and when holding a lock (lock lease time). Operations that modify objects in a global region use automatic locking. In addition, you can manually lock a global region and its entries through `org.apache.geode.cache.Region`. The explicit lock methods provided by the APIs allow you to specify a lock timeout parameter. The lock time out for implicit operations and the lock lease time for implicit and explicit operations are governed by these cache-wide settings:

```
gfsh>alter runtime --lock-timeout=30 --lock-lease=60
```

- `--lock-timeout`. Timeout for object lock requests, specified in seconds. The setting affects automatic locking only, and does not apply to manual locking. The default is 1 minute. If a lock request does not return before the specified timeout period, it is cancelled and returns with a failure.
- `--lock-lease`. Timeout for object lock leases, specified in seconds. The setting affects both automatic locking and manual locking. The default is 2 minutes. Once a lock is obtained, it may remain in force for the lock lease time period before being automatically cleared by the system.

Controlling Socket Use

For peer-to-peer communication, you can manage socket use at the system member level and at the thread level.

The conserve-sockets setting indicates whether application threads share sockets with other threads or use their own sockets for member communication. This setting has no effect on communication between a server and its clients, but it does control the server's communication with its peers or a gateway sender's communication with a gateway receiver. In client/server settings in particular, where there can be a large number of clients for each server, controlling peer-to-peer socket use is an important part of tuning server performance.

You configure conserve-sockets for the member as a whole in `gemfire.properties`. Additionally, you can change the sockets conservation policy for the individual thread through the API.

When conserve-sockets is set to false, each application thread uses a dedicated thread to send to each of its peers and a dedicated thread to receive from each peer. Disabling socket conservation requires more system resources, but can potentially improve performance by removing socket contention between threads and optimizing distributed ACK operations. For distributed regions, the put operation, and destroy and invalidate for regions and entries, can all be optimized with conserve-sockets set to false. For partitioned regions, setting conserve-sockets to false can improve general throughput.

Note: When you have transactions operating on EMPTY, NORMAL or PARTITION regions, make sure that `conserve-sockets` is set to false to avoid distributed deadlocks.

You can override the `conserve-sockets` setting for individual threads. These methods are in `org.apache.geode.distributed.DistributedSystem`:

- `setThreadsSocketPolicy`. Sets the calling thread's individual socket policy, overriding the policy set for the application as a whole. If set to true, the calling thread shares socket connections with other threads. If false, the calling thread has its own sockets.
- `releaseThreadsSockets`. Frees any sockets held by the calling thread. Threads hold their own sockets only when conserve-sockets is false. Threads holding their own sockets can call this method to avoid holding the sockets until the socket-lease-time has expired.

A typical implementation might set conserve-sockets to true at the application level and then override the setting for the specific application threads that perform the bulk of the distributed operations. The example below shows an implementation of the two API calls in a thread that performs benchmark tests. The example assumes the class implements Runnable. Note that the invocation, setThreadsSocketPolicy(false), is only meaningful if conserve-sockets is set to true at the application level.

```
public void run() {
    DistributedSystem.setThreadsSocketPolicy(false);
    try {
        // do your benchmark work
    } finally {
        DistributedSystem.releaseThreadsSockets();
    }
}
```

Management of Slow Receivers

You have several options for handling slow members that receive data distribution. The slow receiver options control only to peer-to-peer communication between distributed regions using TCP/IP. This topic does not apply to client/server or multi-site communication, or to communication using the UDP unicast or IP multicast protocols.

Most of the options for handling slow members are related to on-site configuration during system integration and tuning. For this information, see Slow Receivers with TCP/IP.

Slowing is more likely to occur when applications run many threads, send large messages (due to large entry values), or have a mix of region configurations.

Note: If you are experiencing slow performance and are sending large objects (multiple megabytes), before implementing these slow receiver options make sure your socket buffer sizes are large enough for the objects you distribute. The socket buffer size is set using gemfire.socket-buffer-size.

By default, distribution between system members is performed synchronously. With synchronous communication, when one member is slow to receive, it can cause its producer members to slow down as well. This can lead to general performance problems in the cluster.

The specifications for handling slow receipt primarily affect how your members manage distribution for regions with distributed-no-ack scope, but it can affect other distributed scopes as well. If no regions have distributed-no-ack scope, this mechanism is unlikely to kick in at all. When slow receipt handling does kick in, however, it affects all distribution between the producer and consumer, regardless of scope. Partitioned regions ignore the scope attribute, but for the purposes of this discussion you should think of them as having an implicit distributed-ack scope.

Configuration Options

The slow receiver options are set in the producer member's region attribute, enable-async-conflation, and in the consumer member's async* `gemfire.properties` settings.

Delivery Retries

If the receiver fails to receive a message, the sender continues to attempt to deliver the message as long as the receiving member is still in the cluster. During the retry cycle, throws warnings that include this string:

```
will reattempt
```

The warnings are followed by an info message when the delivery finally succeeds.

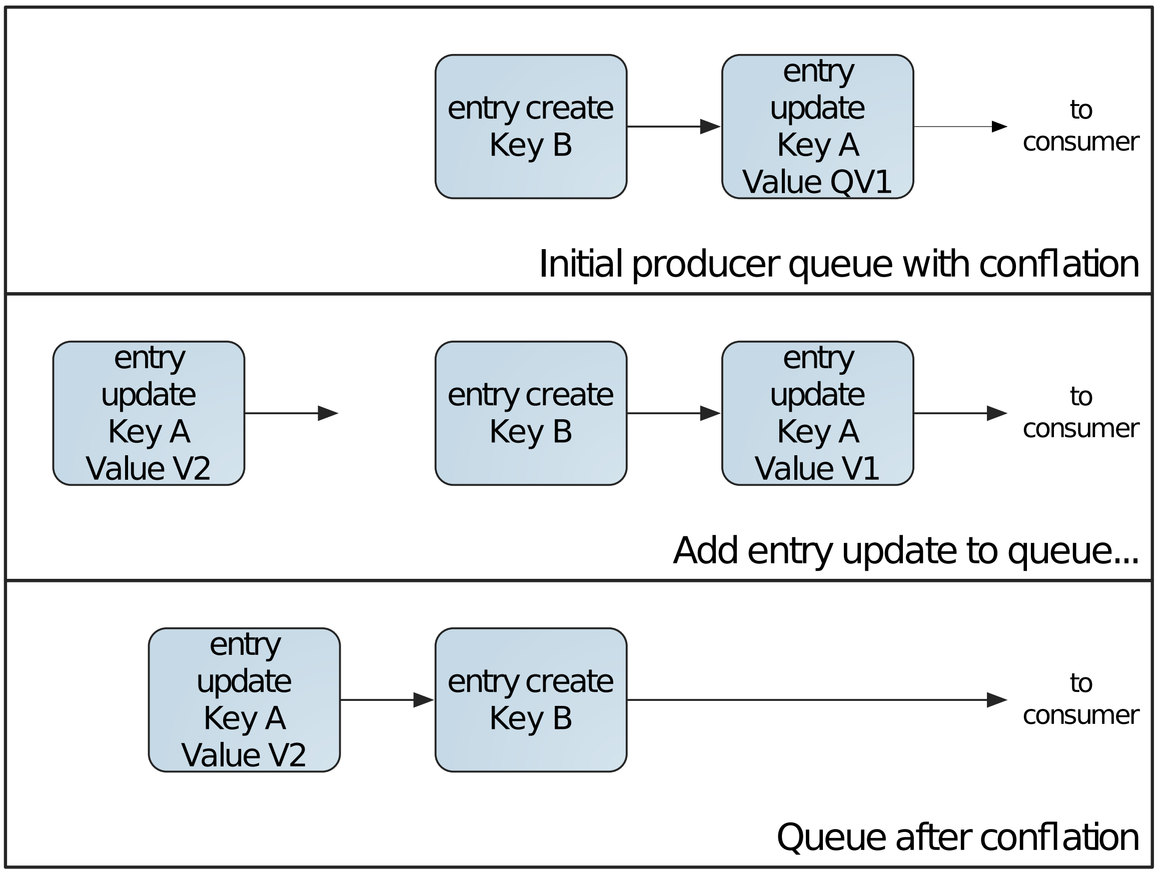
Asynchronous Queueing For Slow Receivers

Your consumer members can be configured so that their producers switch to asynchronous messaging if the consumers are slow to respond to cache message distribution.

When a producer switches, it creates a queue to hold and manage that consumer's cache messages. When the queue empties, the producer switches back to synchronous messaging for the consumer. The settings that cause the producers to switch are specified on the consumer side in `gemfire.properties` file settings.

If you configure your consumers for slow receipt queuing, and your region scope is distributed-no-ack, you can also configure the producer to conflate entry update messages in its queues. This configuration option is set as the region attribute enable-async-conflation. By default distributed-no-ack entry update messages are not conflated.

Depending on the application, conflation can greatly reduce the number of messages the producer needs to send to the consumer. With conflation, when an entry update is added to the queue, if the last operation queued for that key is also an update operation, the previously enqueued update is removed, leaving only the latest update to be sent to the consumer. Only entry update messages originating in a region with distributed-no-ack scope are conflated. Region operations and entry operations other than updates are not conflated.


entry create Key B
entry update Key A Value QV1
to consumer
Initial producer queue with conflation
entry update Key A Value V2
entry create Key B
entry update Key A Value V1
to consumer
Add entry update to queue...
entry update Key A Value V2
entry create Key B
to consumer
Queue after conflation

Some conflation may not occur because entry updates are sent to the consumer before they can be conflated. For this example, assume no messages are sent while the update for Key A is added.

Note: This method of conflation behaves the same as server-to-client conflation.

You can enable queue conflation on a region-by-region basis. You should always enable it unless it is incompatible with your application needs. Conflation reduces the amount of data queued and distributed.

These are reasons why conflation might not work for your application:

- With conflation, earlier entry updates are removed from the queue and replaced by updates sent later in the queue. This is problematic for applications that depend on a specific ordering of entry modifications. For example, if your receiver has a CacheListener that needs to know about every state change, you should deactivate conflation.
- If your queue remains in use for a significant period and you have entries that are updated frequently, you could have a series of update message replacements resulting in a notable delay in the arrival of any update for some entries. Imagine that update 1, before it is sent, is removed in favor of a later update 2. Then, before update 2 can be sent, it is removed in favor of update 3, and so on. This could result in unacceptably stale data on the receiver.

Increasing the Ratio of Cache Hits

The more frequently a get fails to find a valid value in the first cache and has to try a second cache, the more the overall performance is affected.

A common cause of misses is expiration or eviction of the entry. If you have a region's entry expiration or eviction enabled, monitor the region and entry statistics.

If you see a high ratio of misses to hits on the entries, consider increasing the expiration times or the maximum values for eviction, if possible. See Eviction for more information.

System Member Performance

You can modify some configuration parameters to improve system member performance.

Before doing so, you should understand Basic Configuration and Programming.

- **Member Properties**

  Several performance-related properties apply to a cache server or application that connects to the cluster.

- **JVM Memory Settings and System Performance**

  You configure JVM memory settings for the Java application by adding parameters to the java invocation. For the cache server, you add them to the command-line parameters for the gfsh `start server` command.

- **Garbage Collection and System Performance**

  If your application exhibits unacceptably high latencies, you might improve performance by modifying your JVM's garbage collection behavior.

Member Properties

Several performance-related properties apply to a cache server or application that connects to the cluster.

- **statistic-sampling-enabled**.Turning off statistics sampling saves resources, but it also takes away potentially valuable information for ongoing system tuning and unexpected system problems. If LRU eviction is configured, then statistics sampling must be on.
- **statistic-sample-rate**. Increasing the sample rate for statistics reduces system resource use while still providing some statistics for system tuning and failure analysis.
- **log-level**. As with the statistic sample rate, lowering this setting reduces system resource consumption. See Logging.

JVM Memory Settings and System Performance

You configure JVM memory settings for the Java application by adding parameters to the java invocation. For the cache server, you add them to the command-line parameters for the gfsh `start server` command.

- JVM heap size—Your JVM may require more memory than is allocated by default. For example, you may need to increase heap size for an application that stores a lot of data. You can set a maximum size and an initial size, so if you know you will be using the maximum (or close to it) for the life of the member, you can speed memory allocation time by setting the initial size to the maximum. This sets both the maximum and initial memory sizes to 1024 megabytes for a Java application:

```
-Xmx1024m -Xms1024m
```

Properties can be passed to the cache server on the `gfsh` command line:

```
gfsh>start server --name=server-name --J=-Xmx1024m --J=-Xms1024m
```

- MaxDirectMemorySize—The JVM has a kind of memory called direct memory, which is distinct from normal JVM heap memory, that can run out. You can increase the direct buffer memory either by increasing the maximum heap size (see previous JVM Heap Size), which increases both the maximum heap and the maximum direct memory, or by only increasing the maximum direct memory using -XX:MaxDirectMemorySize. The following parameter added to the Java application startup increases the maximum direct memory size to 256 megabytes:

  ```
  -XX:MaxDirectMemorySize=256M
  ```

  The same effect for the cache server:

  ```
  gfsh>start server --name=server-name --J=-XX:MaxDirectMemorySize=256M
  ```

- JVM stack size—Each thread in a Java application has its own stack. The stack is used to hold return addresses, arguments to functions and method calls, and so on. Since VMware GemFire is a highly multi-threaded system, at any given point in time there are multiple thread pools and threads that are in use. The default stack size setting for a thread in Java is 1MB. Stack size has to be allocated in contiguous blocks and if the machine is being used actively and there are many threads running in the system (Task Manager shows the number of active threads), you may encounter an `OutOfMemory error: unable to create new native thread`, even though your process has enough available heap. If this happens, consider reducing the stack size requirement for threads on the cache server. The following parameter added to the Java application startup limits the maximum size of the stack.

  ```
  -Xss384k
  ```

  In particular, we recommend starting the cache servers with a stack size of 384k or 512k in such cases. For example:

  ```
  gfsh>start server --name=server-name --J=-Xss384k

  gfsh>start server --name=server-name --J=-Xss512k
  ```

- Off-heap memory size—For applications that use off-heap memory, specifies how much off-heap memory to allocate. Setting `off-heap-memory-size` is prerequisite to enabling the off-heap capability for individual regions. For example:

  ```
  gfsh>start server --name=server-name --off-heap-memory-size=200G
  ```

  See Using Off-heap Memory for additional considerations regarding this parameter.

- Lock memory—On Linux systems, you can prevent heap and off-heap memory from being paged out by setting the `lock-memory` parameter to `true`. For example:

```
gfsh>start server --name=server-name --off-heap-memory-size=200G --lock-memory=
true
```

See Locking Memory for additional considerations regarding this parameter.

Garbage Collection and System Performance

If your application exhibits unacceptably high latencies, you might improve performance by modifying your JVM's garbage collection behavior.

Garbage collection, while necessary, introduces latency into your system by consuming resources that would otherwise be available to your application. You can reduce the impact of garbage collection in two ways:

- Optimize garbage collection in the JVM heap.
- Reduce the amount of data exposed to garbage collection by storing values in off-heap memory.

Note: Garbage collection tuning options depend on the JVM you are using. Suggestions given here apply to the Sun HotSpot JVM. If you use a different JVM, check with your vendor to see if these or comparable options are available to you.

Note: Modifications to garbage collection sometimes produce unexpected results. Always test your system before and after making changes to verify that the system's performance has improved.

Optimizing Garbage Collection

The two options suggested here are likely to expedite garbage collecting activities by introducing parallelism and by focusing on the data that is most likely to be ready for cleanup. The first parameter causes the garbage collector to run concurrent to your application processes. The second parameter causes it to run multiple, parallel threads for the "young generation" garbage collection (that is, garbage collection performed on the most recent objects in memory—where the greatest benefits are expected):

```
-XX:+UseConcMarkSweepGC -XX:+UseParNewGC
```

For applications, if you are using remote method invocation (RMI) Java APIs, you might also be able to reduce latency by disabling explicit calls to the garbage collector. The RMI internals automatically invoke garbage collection every sixty seconds to ensure that objects introduced by RMI activities are cleaned up. Your JVM may be able to handle these additional garbage collection needs. If so, your application may run faster with explicit garbage collection disabled. You can try adding the following command-line parameter to your application invocation and test to see if your garbage collector is able to keep up with demand:

```
-XX:+DisableExplicitGC
```

Using Off-heap Memory

You can improve the performance of some applications by storing data values in off-heap memory. Certain objects, such as keys, must remain in the JVM heap. See Managing Off-Heap Memory for more information.

Slow Receivers with TCP/IP

You have several options for preventing situations that can cause slow receivers of data distributions. The slow receiver options control only peer-to-peer communication using TCP/IP. This discussion does not apply to client/server or multi-site communication, or to communication using the UDP unicast or multicast protocols.

Before you begin, you should understand VMware GemFire Basic Configuration and Programming.

- **Preventing Slow Receivers**

  During system integration, you can identify and eliminate potential causes of slow receivers in peer-to-peer communication.

- **Managing Slow Receivers**

  If the receiver fails to receive a message, the sender continues to attempt to deliver the message as long as the receiving member is still in the cluster.

Preventing Slow Receivers

During system integration, you can identify and eliminate potential causes of slow receivers in peer-to-peer communication.

Work with your network administrator to eliminate any problems you identify.

Slowing is more likely to occur when applications run many threads, send large messages (due to large entry values), or have a mix of region configurations. The problem can also arise from message delivery retries caused by intermittent connection problems.

Host Resources

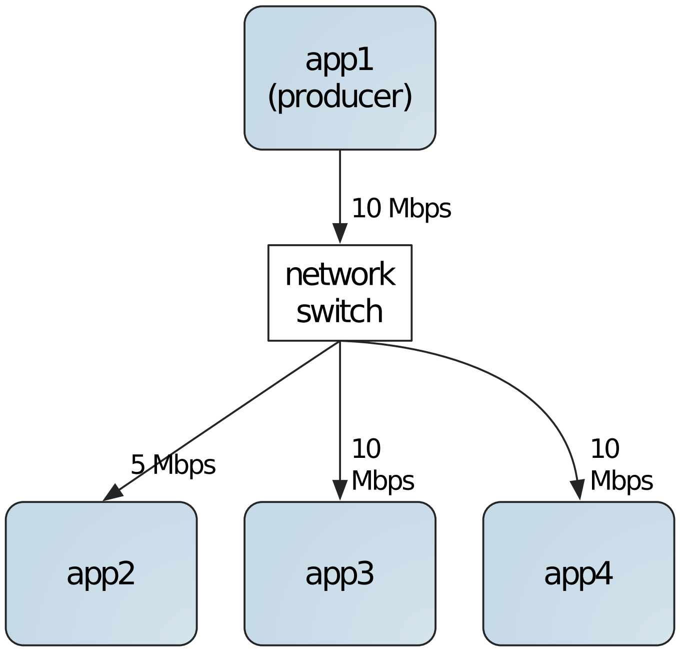
Make sure that the machines that run VMware GemFire members have enough CPU available to them. Do not run any other heavyweight processes on the same machine.

The machines that host VMware GemFire application and cache server processes should have comparable computing power and memory capacity. Otherwise, members on the less powerful machines tend to have trouble keeping up with the rest of the group.

Network Capacity

Eliminate congested areas on the network by rebalancing the traffic load. Work with your network administrator to identify and eliminate traffic bottlenecks, whether caused by the architecture of the distributed VMware GemFire system or by contention between the VMware GemFire traffic and other traffic on your network. Consider whether more subnets are needed to separate the VMware GemFire administrative traffic from VMware GemFire data transport and to separate all the VMware GemFire traffic from the rest of your network load.

The network connections between hosts need to have equal bandwidth. If not, you can end up with a configuration like the multicast example in the following figure, which creates conflicts among the members. For example, if app1 sends out data at 7Mbps, app3 and app4 would be fine, but app2 would miss some data. In that case, app2 contacts app1 on the TCP channel and sends a



log message that it's dropping data.

Plan for Growth

Upgrade the infrastructure to the level required for acceptable performance. Analyze the expected VMware GemFire traffic in comparison to the network's capacity. Build in extra capacity for growth and high-traffic spikes. Similarly, evaluate whether the machines that host VMware GemFire application and cache server processes can handle the expected load.

Managing Slow Receivers

If the receiver fails to receive a message, the sender continues to attempt to deliver the message as long as the receiving member is still in the cluster.

During the retry cycle, VMware GemFire throws warnings that include this string:

```
will reattempt
```

The warnings are followed by an informational message when the delivery finally succeeds.

For distributed regions, the scope of a region determines whether distribution acknowledgments and distributed synchronization are required. Partitioned regions ignore the scope attribute, but for the purposes of this discussion you should think of them as having an implicit distributed-ack scope.

By default, distribution between system members is performed synchronously. With synchronous communication, when one member is slow to receive, it can cause its producers to slow down as well. This, of course, can lead to general performance problems in the cluster.

If you are experiencing slow performance and are sending large objects (multiple megabytes), before implementing these slow receiver options make sure your socket buffer sizes are appropriate for the size of the objects you distribute. The socket buffer size is set using socket-buffer-size in the `gemfire.properties` file.

Managing Slow distributed-no-ack Receivers

You can configure your consumer members so their messages are queued separately when they are slow to respond. The queueing happens in the producer members when the producers detect slow receipt and allows the producers to keep sending to other consumers at a normal rate. Any member that receives data distribution can be configured as described in this section.

The specifications for handling slow receipt primarily affect how your members manage distribution for regions with distributed-no-ack scope, where distribution is asynchronous, but the specifications can affect other distributed scopes as well. If no regions have distributed-no-ack scope, the mechanism is unlikely to kick in at all. When slow receipt handling does kick in, however, it affects all distribution between the producer and that consumer, regardless of scope.

Note: These slow receiver options are deactivated in systems using SSL. See SSL.

Each consumer member determines how its own slow behavior is to be handled by its producers. The settings are specified as distributed system connection properties. This section describes the settings and lists the associated properties.

- async-distribution-timeout—The distribution timeout specifies how long producers are to wait for the consumer to respond to synchronous messaging before switching to asynchronous messaging with that consumer. When a producer switches to asynchronous messaging, it creates a queue for that consumer’s messages and a separate thread to handle the communication. When the queue empties, the producer automatically switches back to synchronous communication with the consumer. These settings affect how long your producer’s cache operations might block. The sum of the timeouts for all consumers is the longest time your producer might block on a cache operation.
- async-queue-timeout—The queue timeout sets a limit on the length of time the asynchronous messaging queue can exist without a successful distribution to the slow receiver. When the timeout is reached, the producer asks the consumer to leave the cluster.
- async-max-queue-size—The maximum queue size limits the amount of memory the asynchronous messaging queue can consume. When the maximum is reached, the producer asks the consumer to leave the cluster.

Configuring Async Queue Conflation

When the scope is distributed-no-ack scope, you can configure the producer to conflate entry update messages in its queues, which may further speed communication. By default, distributed-no-ack entry update messages are not conflated. The configuration is set in the producer at the region level.

Forcing the Slow Receiver to Disconnect

If either of the queue timeout or maximum queue size limits is reached, the producer sends the consumer a high-priority message (on a different TCP connection than the connection used for cache messaging) telling it to disconnect from the cluster. This prevents growing memory consumption by the other processes that are queuing changes for the slow receiver while they wait for that receiver to catch up. It also allows the slow member to start fresh, possibly clearing up the issues that were causing it to run slowly.

When a producer gives up on a slow receiver, it logs one of these types of warnings:

- Blocked for time ms which is longer than the max of asyncQueueTimeout ms so asking slow receiver slow_receiver_ID to disconnect.

- Queued bytes exceed max of asyncMaxQueueSize so asking slow receiver slow_receiver_ID to disconnect.

When a process disconnects after receiving a request to do so by a producer, it logs a warning message of this type:

- Disconnect forced by producer because we were too slow.

These messages only appear in your logs if logging is enabled and the log level is set to a level that includes warning (which it does by default). See Logging.

If your consumer is unable to receive even high priority messages, only the producer's warnings will appear in the logs. If you see only producer warnings, you can restart the consumer process. Otherwise, the VMware GemFire failure detection code will eventually cause the member to leave the cluster on its own.

Use Cases

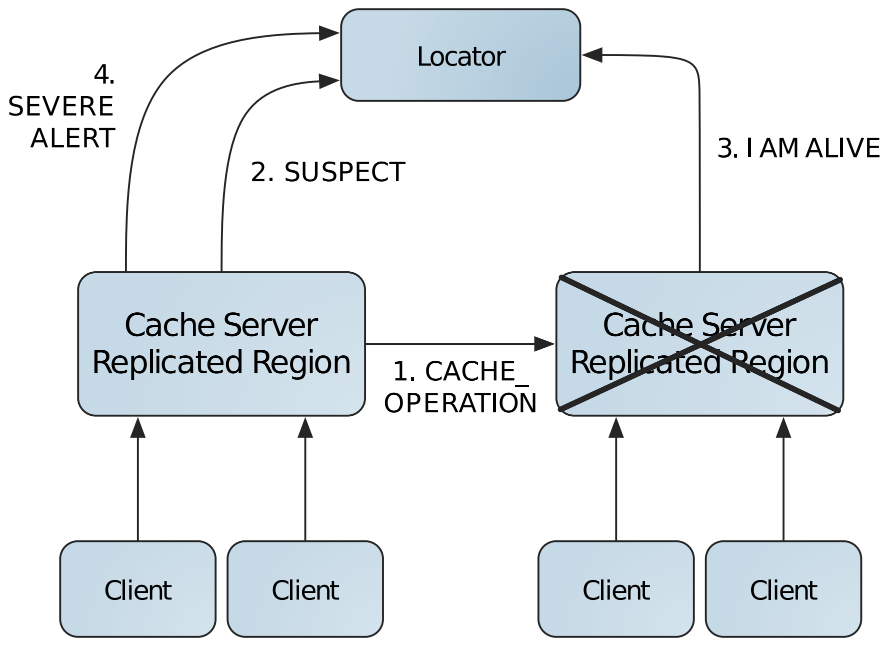
These are the main use cases for the slow receiver specifications:

- Message bursts—With message bursts, the socket buffer can overflow and cause the producer to block. To keep from blocking, first make sure your socket buffer is large enough to handle a normal number of messages (using the socket-buffer-size property), then set the async distribution timeout to 1. With this very low distribution timeout, when your socket buffer does fill up, the producer quickly switches to async queueing. Use the distribution statistics, asyncQueueTimeoutExceeded and asyncQueueSizeExceeded, to make sure your queue settings are high enough to avoid forcing unwanted disconnects during message bursts.
- Unhealthy or dead members—When members are dead or very unhealthy, they may not be able to communicate with other members. The slow receiver specifications allow you to force crippled members to disconnect, freeing up resources and possibly allowing the members to restart fresh. To configure for this, set the distribution timeout high (one minute), and set the queue timeout low. This is the best way to avoid queueing for momentary slowness, while still quickly telling very unhealthy members to leave the cluster.
- Combination message bursts and unhealthy members—To configure for both of the above situations, set the distribution timeout low and the queue timeout high, as for the message bursts scenario.

Managing Slow distributed-ack Receivers

When using a distribution scope other than distributed-no-ack, alerts are issued for slow receivers. A member that isn't responding to messages may be sick, slow, or missing. Sick or slow members are detected in message transmission and reply-wait processing code, triggering a warning alert first. If a member still isn't responding, a severe warning alert is issued, indicating that the member may be disconnected from the cluster. This alert sequence is enabled by setting the ack-wait-threshold and the ack-severe-alert-threshold to some number of seconds.

When ack-severe-alert-threshold is set, regions are configured to use ether distributed-ack or global scope, or use the partition data policy. VMware GemFire will wait for a total of ack-wait-threshold seconds for a response to a cache operation, then it logs a warning alert ("Membership: requesting removal of entry(#). Disconnected as a slow-receiver"). After waiting an additional ack-severe-alert-threshold seconds after the first threshold is reached, the system also informs the failure detection mechanism that the receiver is suspect and may be disconnected, as shown in the following figure.



The events occur in this order:

1. CACHE_OPERATION: Transmission of cache operation is initiated.
2. SUSPECT: Identified as a suspect by ack-wait-threshold, which is the maximum time to wait for an acknowledge before initiating failure detection.
3. I AM ALIVE: Notification to the system in response to failure detection queries, if the process is still alive. A new membership view is sent to all members if the suspect process fails to answer with I AM ALIVE.
4. SEVERE ALERT: The result of ack-severe-wait-threshold elapsing without receiving a reply.

When a member fails suspect processing, its cache is closed and its CacheListeners are notified with the afterRegionDestroyed notification. The RegionEvent passed with this notification has a CACHE_CLOSED operation and a FORCED_DISCONNECT operation, as shown in the FORCED_DISCONNECT example.

```
public static final Operation FORCED_DISCONNECT
= new Operation("FORCED_DISCONNECT",
        true, // isLocal
        true, // isRegion
        OP_TYPE_DESTROY,
        OP_DETAILS_NONE
        );
```

A cache closes due to being expelled from the cluster by other members. Typically, this happens when a member becomes unresponsive and does not respond to heartbeat requests within the member-timeout period, or when ack-severe-alert-threshold has expired without a response from the member.

Note: This is marked as a region operation.

Other members see the normal membership notifications for the departing member. For instance, RegionMembershipListeners receive the afterRemoteRegionCrashed notification, and SystemMembershipListeners receive the memberCrashed notification.

Slow distributed-ack Messages

In systems with distributed-ack regions, a sudden large number of distributed-no-ack operations can cause distributed-ack operations to take a long time to complete.

The `distributed-no-ack` operations can come from anywhere. They may be updates to `distributed-no-ack` regions or they may be other `distributed-no-ack` operations, like destroys, performed on any region in the cache, including the `distributed-ack` regions.

The main reasons why a large number of `distributed-no-ack` messages may delay `distributed-ack` operations are:

- For any single socket connection, all operations are executed serially. If there are any other operations buffered for transmission when a `distributed-ack` is sent, the `distributed-ack` operation must wait to get to the front of the line before being transmitted. Of course, the operation's calling process is also left waiting.
- The `distributed-no-ack` messages are buffered by their threads before transmission. If many messages are buffered and then sent to the socket at once, the line for transmission might be very long.

You can take these steps to reduce the impact of this problem:

1. If you're using TCP, check whether you have socket conservation enabled for your members. It is configured by setting the VMware GemFire property `conserve-sockets` to true. If enabled, each application's threads will share sockets unless you override the setting at the thread level. Work with your application programmers to see whether you might disable sharing entirely or at least for the threads that perform `distributed-ack` operations. These include operations on `distributed-ack` regions and also `netSearches` performed on regions of any distributed scope. (Note: `netSearch` is only performed on regions with a data-policy of empty, normal and preloaded.) If you give each thread that performs `distributed-ack` operations its own socket, you effectively let it scoot to the front of the line ahead of the `distributed-no-ack` operations that are being performed by other threads. The thread-level override is done by calling the `DistributedSystem.setThreadsSocketPolicy(false)` method.
2. Reduce your buffer sizes to slow down the distributed-no-ack operations. These changes slow down the threads performing distributed-no-ack operations and allow the thread doing the distributed-ack operations to be sent in a more timely manner.
    - If you're using UDP (you either have multicast enabled regions or have set `disable-tcp` to true in gemfire.properties), consider reducing the byteAllowance of mcast-flow-control to something smaller than the default of 3.5 megabytes.
    - If you're using TCP/IP, reduce the `socket-buffer-size` in gemfire.properties.

Socket Communication

VMware GemFire processes communicate using TCP/IP and UDP unicast and multicast protocols. In all cases, communication uses sockets that you can tune to optimize performance.

The adjustments you make to tune your VMware GemFire communication may run up against operating system limits. If this happens, check with your system administrator about adjusting the operating system settings.

All of the settings discussed here are listed as `gemfire.properties` and `cache.xml` settings. They can also be configured through the API and some can be configured at the command line. Before you begin, you should understand VMware GemFire Basic Configuration and Programming.

- **Setting Socket Buffer Sizes**

  When you determine buffer size settings, you try to strike a balance between communication needs and other processing.

- **Ephemeral TCP Port Limits**

  By default, Windows' ephemeral ports are within the range 1024-4999, inclusive.You can increase the range.

- **Making Sure You Have Enough Sockets**

  The number of sockets available to your applications is governed by operating system limits.

- **TCP/IP KeepAlive Configuration**

  VMware GemFire supports TCP KeepAlive to prevent socket connections from being timed out.

- **TCP/IP Peer-to-Peer Handshake Timeouts**

  You can alleviate connection handshake timeouts for TCP/IP connections by increasing the connection handshake timeout interval with the system property p2p.handshakeTimeoutMs.

Setting Socket Buffer Sizes

When you determine buffer size settings, you must strike a balance between communication needs and other processing.

Larger socket buffers allow your members to distribute data and events more quickly, but they also take memory away from other things. If you store very large data objects in your cache, finding the right sizing for your buffers while leaving enough memory for the cached data can become critical to system performance.

Ideally, you should have buffers large enough for the distribution of any single data object so you don't get message fragmentation, which lowers performance. Your buffers should be at least as large as your largest stored objects and their keys plus some overhead for message headers. The overhead varies depending on who is sending and receiving, but 100 bytes should be sufficient. You can also look at the statistics for the communication between your processes to see how many bytes are being sent and received.

If you see performance problems and logging messages indicating blocked writers, increasing your buffer sizes may help.

This table lists the settings for the various member relationships and protocols, and tells where to set them.

Protocol / Area Affected	Configuration Location	Property Name
TCP / IP	---	---
Peer-to-peer send/receive	gemfire.properties	socket-buffer-size
Client send/receive	cache.xml <pool>	socket-buffer-size
Server send/receive	`gfsh start server` or cache.xml <CacheServer>	socket-buffer-size
UDP Multicast	---	---
Peer-to-peer send	gemfire.properties	mcast-send-buffer-size
Peer-to-peer receive	gemfire.properties	mcast-recv-buffer-size
UDP Unicast	---	---
Peer-to-peer send	gemfire.properties	udp-send-buffer-size
Peer-to-peer receive	gemfire.properties	udp-recv-buffer-size

TCP/IP Buffer Sizes

If possible, your TCP/IP buffer size settings should match across your VMware GemFire installation. At a minimum, follow the guidelines listed here.

- **Peer-to-peer**. The socket-buffer-size setting in `gemfire.properties` should be the same throughout your cluster.
- **Client/server**. The client's pool socket-buffer size-should match the setting for the servers the pool uses, as in these example `cache.xml` snippets:

```
Client Socket Buffer Size cache.xml Configuration:
<pool>name="PoolA" server-group="dataSetA" socket-buffer-size="42000"...

Server Socket Buffer Size cache.xml Configuration:
<cache-server port="40404" socket-buffer-size="42000">
    <group>dataSetA</group>
</cache-server>
```

UDP Multicast and Unicast Buffer Sizes

With UDP communication, one receiver can have many senders sending to it at once. To accommodate all of the transmissions, the receiving buffer should be larger than the sum of the sending buffers. If you have a system with at most five members running at any time, in which all members update their data regions, you would set the receiving buffer to at least five times the size of the sending buffer. If you have a system with producer and consumer members, where only two producer members ever run at once, the receiving buffer sizes should be set at over two times the sending buffer sizes, as shown in this example:

```
mcast-send-buffer-size=42000
mcast-recv-buffer-size=90000
udp-send-buffer-size=42000
udp-recv-buffer-size=90000
```

Operating System Limits

Your operating system sets limits on the buffer sizes it allows. If you request a size larger than the allowed, you may get warnings or exceptions about the setting during startup. These are two examples of the type of message you may see:

```
[warning 2008/06/24 16:32:20.286 PDT CacheRunner <main> tid=0x1]
requested multicast send buffer size of 9999999 but got 262144: see
system administration guide for how to adjust your OS

Exception in thread "main" java.lang.IllegalArgumentException: Could not
set "socket-buffer-size" to "99262144" because its value can not be
greater than "20000000".
```

If you think you are requesting more space for your buffer sizes than your system allows, check with your system administrator about adjusting the operating system limits.

Ephemeral TCP Port Limits

By default, Windows' ephemeral ports are within the range 1024-4999, inclusive. You can increase the range.

If you are repeatedly receiving the following exception:

```
java.net.BindException: Address already in use: connect
```

and if your system is experiencing a high degree of network activity, such as numerous short-lived client connections, this could be related to a limit on the number of ephemeral TCP ports. While this issue could occur with other operating systems, typically, it is seen only with Windows due to a low default limit.

Perform this procedure to increase the limit:

1. Open the Windows Registry Editor.
2. Navigate to the following key:

   ```
   HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Tcpip\Parameter
   ```

3. From the Edit menu, click New, and then add the following registry entry:

   ```
   Value Name: MaxUserPort
   Value Type: DWORD
   Value data: 36863
   ```

4. Exit the Registry Editor, and then restart the computer.

This affects all versions of the Windows operating system.

Note for UDP on Unix Systems

Unix systems have a default maximum socket buffer size for receiving UDP multicast and unicast transmissions that is lower than the default settings for `mcast-recv-buffer-size` and `udp-recv-buffer-size`. To achieve high-volume multicast messaging, you should increase the maximum Unix buffer size to at least one megabyte.

Making Sure You Have Enough Sockets

The number of sockets available to your applications is governed by operating system limits.

Sockets use file descriptors and the operating system's view of your application's socket use is expressed in terms of file descriptors. There are two limits, one on the maximum descriptors available to a single application and the other on the total number of descriptors available in the system. If you get error messages telling you that you have too many files open, you might be hitting the operating system limits with your use of sockets. Your system administrator might be able to increase the system limits so that you have more available. You can also tune your members to use fewer sockets for their outgoing connections. This section discusses socket use in VMware GemFire and ways to limit socket consumption in your VMware GemFire members.

Socket Sharing

You can configure socket sharing for peer-to-peer:

- **Peer-to-peer**. You can configure whether your members share sockets both at the application level and at the thread level. To enable sharing at the application level, set the `gemfire.properties` property `conserve-sockets` to `true`. To achieve maximum throughput, however, we recommend that you use the default value of `false`.

  At the thread level, developers can override this setting by using the DistributedSystem API method `setThreadsSocketPolicy`. You might want to enable socket sharing at the application level and then have threads that do a lot of cache work take sole ownership of their sockets. Make sure to program these threads to release their sockets as soon as possible using the `releaseThreadsSockets` method, rather than waiting for a timeout or thread death.

Socket Lease Time

You can force the release of an idle socket connection for peer-to-peer and client-to-server connections:

- **Peer-to-peer**. For peer-to-peer threads that do not share sockets, you can use the `socket-lease-time` to make sure that no socket sits idle for too long. When a socket that belongs to an individual thread remains unused for this time period, the system automatically closes that socket. The next time the thread needs a socket, it creates a new socket.
- **Client**. For client connections, you can affect the same lease-time behavior by setting the pool `idle-timeout`.

Calculating Connection Requirements

Each type of member has its own connection requirements. Clients need connections to their servers, peers need connections to peers, and so on. Many members have compound roles. Use these guidelines to figure each member's socket needs and to calculate the combined needs of members that run on a single host system.

A member's socket use is governed by a number of factors, including:

- How many peer members it connects to
- How many threads it has that update the cache and whether the threads share sockets
- Whether it is a server or a client,
- How many connections come in from other processes

The socket requirements described here are worst-case. Generally, it is not practical to calculate exact socket use for your applications. Socket use varies depending on a number of factors including how many members are running, what their threads are doing, and whether threads share sockets.

To calculate any member's socket requirements, add up the requirements for every category that applies to the member. For example, a cache server running in a cluster with clients connected to it has both peer-to-peer and server socket requirements.

Peer-to-Peer Socket Requirements Per Member

Every member of a cluster maintains two outgoing and two incoming connections to every peer. If threads share sockets, these fixed sockets are the sockets they share.

For every thread that does not share sockets, additional sockets, one in and one out, are added for each peer. This affects not only the member's socket count, but the socket count for every member the member thread connects to.

In this table:

- M is the total number of members in the cluster.
- T is the number of threads in a member that own their own sockets and do not share.

Peer Member Socket Description	Number Used
Membership failure detection	2
Listener for incoming peer connections (server P2P)	1
Shared sockets (2 in and 2 out) Threads that share sockets use these.	4 * (M-1)
This member's thread-owned sockets (1 in and 1 out for each thread, for each peer member).	(T * 2) * (M-1)

Peer Member Socket Description	Number Used
Other member's thread-owned sockets that connect to this member (1 in and 1 out for each). Note that this might include server threads if any of the other members are servers (see Server).	Summation over (M-1) other members of (T*2)

Note: The threads servicing client requests add to the total count of thread-owned sockets both for this member connecting to its peers and for peers that connect to this member.

Server Socket Requirements Per Server

Servers use one connection for each incoming client connection. By default, each connection is serviced by a server thread. These threads that service client requests communicate with the rest of the servers to satisfy the requests and distributed update operations. Each of these threads uses its own thread-owned sockets for peer-to-peer communication. So this adds to the server's group of thread-owned sockets.

The thread and connection count in the server may be limited by server configuration settings. These are `max-connections` and `max-threads` settings in the <cache-server> element of the `cache.xml`. These settings limit the number of connections the server accepts and the maximum number of threads that can service client requests. Both of these limit the server's overall connection requirements:

- When the connection limit is reached, the server refuses additional connections. This limits the number of connections the server uses for clients.
- When the thread limit is reached, threads start servicing multiple connections. This does not limit the number of client connections, but does limit the number of peer connections required to service client requests. Each server thread used for clients uses its own sockets, so it requires 2 connections to each of the server's peers. The `max-threads` setting puts a cap on the number of this type of peer connection that your server needs.

The server uses one socket for each incoming client pool connection. If client subscriptions are used, the server creates an additional connection to each client that enables subscriptions.

In this table, M is the total number of members in the cluster.

Server Socket Description	Number Used
Listener for incoming client connections	1
Client pool connections to server	Number of pool connections to this server
Threads servicing client requests (the lesser of the client pool connection count and the server's `max-threads` setting). These connections are to the server's peers.	(2 * number of threads in a server that service client pool connections) * (M-1) These threads do not share sockets.
Subscription connections	2 * number of client subscription connections to this server

With client/server installations, the number of client connections to any single server is undetermined, but VMware GemFire's server load balancing and conditioning keeps the connections fairly evenly distributed among servers.

Servers are peers in their own cluster and have the additional socket requirements as noted in the Peer-to-Peer section above.

Client Socket Requirements per Client

Client connection requirements are compounded by how many pools they use. The use varies according to runtime client connection needs, but will usually have maximum and minimum settings. Look for the <pool> element in the `cache.xml` for the configuration properties.

Client Socket Description	Number Used
Pool connection	summation over the client pools of max-connections
Subscription connections	2 * summation over the client pools of subscription-enabled

If your client acts as a peer in its own cluster, it has the additional socket requirements as noted in the Peer-to-Peer section of this topic.

TCP/IP KeepAlive Configuration

VMware GemFire supports TCP KeepAlive to prevent socket connections from being timed out.

The `gemfire.setTcpKeepAlive` system property prevents connections that appear idle from being timed out (for example, by a firewall.) When configured to true, VMware GemFire enables the SO_KEEPALIVE option for individual sockets. This operating system-level setting allows the socket to send verification checks (ACK requests) to remote systems in order to determine whether or not to keep the socket connection alive.

Note: The time intervals for sending the first ACK KeepAlive request, the subsequent ACK requests and the number of requests to send before closing the socket is configured on the operating system level.

By default, this system property is set to true.

TCP/IP Peer-to-Peer Handshake Timeouts

You can alleviate connection handshake timeouts for TCP/IP connections by increasing the connection handshake timeout interval with the system property `p2p.handshakeTimeoutMs`.

The default setting is 59000 milliseconds.

This sets the handshake timeout to 75000 milliseconds for a Java application:

```
-Dp2p.handshakeTimeoutMs=75000
```

The properties are passed to the cache server on the `gfsh` command line:

```
gfsh>start server --name=server_name --J=-Dp2p.handshakeTimeoutMs=75000
```

Configuring Sockets in Multi-Site (WAN) Deployments

When you determine buffer size settings, you try to strike a balance between communication needs and other processing.

This table lists the settings for gateway relationships and protocols, and tells where to set them.

Protocol / Area Affected	Configuration Location	Property Name
TCP / IP	---	---
Gateway sender	`gfsh create gateway-sender` or cache.xml <gateway-sender>	socket-buffer-size
Gateway receiver	`gfsh create gateway-receiver` or cache.xml <gateway-receiver>	socket-buffer-size

TCP/IP Buffer Sizes

If possible, your TCP/IP buffer size settings should match across your installation. At a minimum, follow the guidelines listed here.

- **Multisite (WAN)**. In a multi-site installation using gateways, if the link between sites is not tuned for optimum throughput, it could cause messages to back up in the cache queues. If a receiving queue overflows because of inadequate buffer sizes, it will become out of sync with the sender and the receiver will be unaware of the condition.

  The gateway sender's socket-buffer-size attribute should match the gateway receiver's socket-buffer-size attribute for all gateway receivers that the sender connects to, as in these example `cache.xml` snippets:

  ```
  Gateway Sender Socket Buffer Size cache.xml Configuration:

  <gateway-sender id="sender2" parallel="true"
   remote-distributed-system-id="2"
   socket-buffer-size="42000"
   maximum-queue-memory="150"/>

  Gateway Receiver Socket Buffer Size cache.xml Configuration:
  <gateway-receiver start-port="1530" end-port="1551"
   socket-buffer-size="42000"/>
  ```

Note: WAN deployments increase the messaging demands on a VMware GemFire system. To avoid hangs related to WAN messaging, always use the default setting of `conserve-sockets=false` for VMware GemFire members that participate in a WAN deployment.

Multi-site (WAN) Socket Requirements

Each gateway sender and gateway receiver uses a socket to distribute events or to listen for incoming connections from remote sites.

Multi-site Socket Description	Number Used
Listener for incoming connections	summation of the number of gateway-receivers defined for the member
Incoming connection	summation of the total number of remote gateway senders configured to connect to the gateway receiver
Outgoing connection	summation of the number of gateway senders defined for the member

Servers are peers in their own clusters and have the additional socket requirements as noted in the Peer-to-Peer section above.

Member produces SocketTimeoutException

A client, server, gateway sender, or gateway receiver produces a `SocketTimeoutException` when it stops waiting for a response from the other side of the connection and closes the socket. This exception typically happens on the handshake or when establishing a callback connection.

Response:

Increase the default socket timeout setting for the member. This timeout is set separately for the client Pool and for the gateway sender and gateway receiver, either in the `cache.xml` file or through the API. For a client/server configuration, adjust the “read-timeout” value as described in <pool> or use the `org.apache.geode.cache.client.PoolFactory.setReadTimeout` method. For a gateway sender or gateway receiver, see WAN Configuration.

UDP Communication

You can make configuration adjustments to improve multicast and unicast UDP performance of peer-to-peer communication.

You can tune your VMware GemFire UDP messaging to maximize throughput. There are two main tuning goals: to use the largest reasonable datagram packet sizes and to reduce retransmission rates. These actions reduce messaging overhead and overall traffic on your network while still getting your data where it needs to go. VMware GemFire also provides statistics to help you decide when to change your UDP messaging settings.

Before you begin, you should understand VMware GemFire Basic Configuration and Programming. See also the general communication tuning and multicast-specific tuning covered in Socket Communication and Multicast Communication.

UDP Datagram Size

You can change the UDP datagram size with the VMware GemFire property `udp-fragment-size`. This is the maximum packet size for transmission over UDP unicast or multicast sockets. When possible, smaller messages are combined into batches up to the size of this setting.

Most operating systems set a maximum transmission size of 64k for UDP datagrams, so this setting should be kept under 60k to allow for communication headers. Setting the fragment size too high can result in extra network traffic if your network is subject to packet loss, as more data must be resent for each retransmission. If many UDP retransmissions appear in DistributionStats, you maybe achieve better throughput by lowering the fragment size.

UDP Flow Control

UDP protocols typically have a flow-control protocol built into them to keep processes from being overrun by incoming no-ack messages. The VMware GemFire UDP flow-control protocol is a credit based system in which the sender has a maximum number of bytes it can send before getting its byte credit count replenished, or recharged, by its receivers. While its byte credits are too low, the sender waits. The receivers do their best to anticipate the sender's recharge requirements and provide recharges before they are needed. If the sender's credits run too low, it explicitly requests a recharge from its receivers.

This flow-control protocol, which is used for all multicast and unicast no-ack messaging, is configured using a three-part VMware GemFire property `mcast-flow-control`. This property is composed of:

- `byteAllowance`—Determines how many bytes (also referred to as credits) can be sent before receiving a recharge from the receiving processes.
- `rechargeThreshold`—Sets a lower limit on the ratio of the sender's remaining credit to its `byteAllowance`. When the ratio goes below this limit, the receiver automatically sends a recharge. This reduces recharge request messaging from the sender and helps keep the sender from blocking while waiting for recharges.
- `rechargeBlockMs`—Tells the sender how long to wait while needing a recharge before explicitly requesting one.

In a well-tuned system, where consumers of cache events are keeping up with producers, the `byteAllowance` can be set high to limit flow-of-control messaging and pauses. JVM bloat or frequent message retransmissions are an indication that cache events from producers are overrunning consumers.

UDP Retransmission Statistics

VMware GemFire stores retransmission statistics for its senders and receivers. You can use these statistics to help determine whether your flow control and fragment size settings are appropriate for your system.

The retransmission rates are stored in the DistributionStats `ucastRetransmits` and `mcastRetransmits`. For multicast, there is also a receiver-side statistic `mcastRetransmitRequests` that can be used to see which processes aren't keeping up and are requesting retransmissions. There is no comparable way to tell which receivers are having trouble receiving unicast UDP messages.

Multicast Communication

You can make configuration adjustments to improve the UDP multicast performance of peer-to-peer communication in your VMware GemFire system.

Before you begin, you should understand VMware GemFire Basic Configuration and Programming. See also the general communication tuning and UDP tuning covered in Socket Communication and UDP Communication.

- **Provisioning Bandwidth for Multicast**

  Multicast installations require more planning and configuration than TCP installations. With IP multicast, you gain scalability but lose the administrative convenience of TCP.

- **Testing Multicast Speed Limits**

  TCP automatically adjusts its speed to the capability of the processes using it and enforces bandwidth sharing so that every process gets a turn. With multicast, you must determine and explicitly set those limits.

- **Configuring Multicast Speed Limits**

  After you determine the maximum transmission rate, configure and tune your production system.

- **Run-time Considerations for Multicast**

  When you use multicast for messaging and data distribution, you need to understand how the health monitoring setting works and how to control memory use.

- **Troubleshooting the Multicast Tuning Process**

  Several problems may arise during the initial testing and tuning process for multicasting.

Provisioning Bandwidth for Multicast

Multicast installations require more planning and configuration than TCP installations. With IP multicast, you gain scalability but lose the administrative convenience of TCP.

When you install an application that runs over TCP, the network is almost always set up for TCP and other applications are already using it. When you install an application to run over IP multicast it may be the first multicast application on the network.

Multicast is very dependent on the environment in which it runs. Its operation is affected by the network hardware, the network software, the machines, which Geode processes run on which machines, and whether there are any competing applications. You could find that your site has connectivity in TCP but not in multicast because some switches and network cards do not support multicast. Your network could have latent problems that you would never see otherwise. To successfully implement a distributed Geode system using multicast requires the cooperation of both system and network administrators.

Bounded Operation Over Multicast

Group rate control is required for Geode systems to maintain cache coherence. If your application delivers the same data to a group of members, your system tuning effort needs to focus on the slow receivers.

If some of your members have trouble keeping up with the incoming data, the other members in the group may be impacted. At best, slow receivers cause the producer to use buffering, adding latency for the slow receiver and perhaps for all of them. In the worst case, throughput for the group can stop entirely while the producer's CPU, memory and network bandwidth are dedicated to serving the slow receivers.

To address this issue, you can implement a bounded operation policy, which sets boundaries for the producer's operation. The appropriate rate limits are determined through tuning and testing to allow the fastest operation possible while minimizing data loss and latency in the group of consumers. This policy is suited to applications such as financial market data, where high throughput, reliable delivery and network stability are required. With the boundaries set correctly, your producer's traffic cannot cause a network outage.

Multicast protocols typically have a flow control protocol built into them to keep processes from being overrun. The Geode flow control protocol uses the mcast-flow-control property to set producer and consumer boundaries for multicast flow operations. The property provides these three configuration settings:

`byteAllowance`	Number of bytes that can be sent without a recharge.
`rechargeThreshold`	Tells consumers how low the producer's initial to remaining allowance ratio should be before sending a recharge.
`rechargeBlockMs`	Tells the producer how long to wait for a recharge before requesting one.

Testing Multicast Speed Limits

TCP automatically adjusts its speed to the capability of the processes using it and enforces bandwidth sharing so that every process gets a turn. With multicast, you must determine and explicitly set those limits.

Without the proper configuration, multicast delivers its traffic as fast as possible, overrunning the ability of consumers to process the data and locking out other processes that are waiting for the bandwidth. You can tune your multicast and unicast behavior using mcast-flow-control in `gemfire.properties`.

Using Iperf

Iperf is an open-source TCP/UDP performance tool that you can use to find your site's maximum rate for data distribution over multicast. Iperf can be downloaded from web sites such as the National Laboratory for Applied Network Research (NLANR).

Iperf measures maximum bandwidth, allowing you to tune parameters and UDP characteristics. Iperf reports statistics on bandwidth, delay jitter, and datagram loss. On Linux, you can redirect this output to a file; on Windows, use the -o filename parameter.

Run each test for ten minutes to make sure any potential problems have a chance to develop. Use the following command lines to start the sender and receivers.

Sender:

```
iperf -c 192.0.2.0 -u -T 1 -t 100 -i 1 -b 1000000000
```

where:

-c address	Run in client mode and connect to a multicast address
-u	Use UDP
-T #	Multicast time-to-live: number of subnets across which a multicast packet can travel before the routers drop the packet

Note: Do not set the -T parameter above 1 without consulting your network administrator. If this number is too high then the iperf traffic could interfere with production applications or continue out onto the internet.

-t	Length of time to transmit, in seconds
-i	Time between periodic bandwidth reports, in seconds
-b	Sending bandwidth, in bits per second

Receiver:

```
iperf -s -u -B 192.0.2.0 -i 1
```

where:

-s	Run in server mode
-u	Use UDP
-B address	Bind to a multicast address
-i #	Time between periodic bandwidth reports, in seconds

Note: If your VMware GemFire cluster runs across several subnets, start a receiver on each subnet.

In the receiver’s output, look at the Lost/Total Datagrams columns for the number and percentage of lost packets out of the total sent.

Output From Iperf Testing:

```
[    ID] Interval     Transfer    Bandwidth   Jitter  Lost/Total Datagrams
[    3] 0.0- 1.0 sec     129 KBytes  1.0 Mbits/sec  0.778 ms     61/    151 (40%)
[    3] 1.0- 2.0 sec     128 KBytes  1.0 Mbits/sec  0.236 ms     0/  89 (0%)
[    3] 2.0- 3.0 sec     128 KBytes  1.0 Mbits/sec  0.264 ms     0/  89 (0%)
[    3] 3.0- 4.0 sec     128 KBytes  1.0 Mbits/sec  0.248 ms     0/  89 (0%)
[    3] 0.0- 4.3 sec     554 KBytes  1.0 Mbits/sec  0.298 ms     61/    447 (14%)
```

Rerun the test at different bandwidths until you find the maximum useful multicast rate. Start high, then gradually decrease the send rate until the test runs consistently with no packet loss. For example, you might need to run five tests in a row, changing the -b (bits per second) parameter each time until there is no loss:

1. -b 1000000000 (loss)
2. -b 900000000 (no loss)
3. -b 950000000 (no loss)
4. -b 980000000 (a bit of loss)
5. -b 960000000 (no loss)

Enter iperf -h to see all of the command-line options. For more information, see the Iperf user manual.

Configuring Multicast Speed Limits

After you determine the maximum transmission rate, configure and tune your production system.

For best performance, the producer and the consumers should run on different machines and each process should have at least one CPU dedicated to it. The following is a list of configuration changes that can improve multicast performance. Check with your system administrator about changing any of the limits discussed here.

- Increase the default datagram size for systems running Microsoft Windows from 1024 bytes to a value that matches your network's maximum transmission unit (MTU), which is typically 1500 bytes. The higher setting should improve the system's network performance.
- Distribution statistics for stack time probes are deactivated by default to increase multicast performance. To reduce multicast speed, you can enable time statistics by setting the gemfire.`enable-time-statistics` property to true.

  This enables time statistics for a Java application:

  ```
  -Dgemfire.enable-time-statistics=true
  ```

  The time statistics properties are passed to the cache server on the `gfsh` the command line:

  ```
  gfsh>start server --name=server_name --enable-time-statistics=true
  ```

- Monitor the members that receive data for signs of data loss. A few data loss messages can happen normally during region creation. Data loss monitoring can be done by reviewing the GemFire DistributionStats in the statistics archive using the optional Visual Statistics Display (VSD) tool. If the cache regions are configured to require acknowledgment, you could see messages timing out as they wait for a response. After a put into a region, the next operations might report that the entry could not be found. Multicast retransmit requests and unicast retransmits can also be monitored to detect data loss. Even when you see data loss, the cause of the problem may have nothing to do with the network. However, if it happens constantly then you should try testing the flow control rate again
- If necessary, reconfigure all the `gemfire.properties` files and repeat with lower flow control maximum credits until you find the maximum useful rate for your installation.

- Slow system performance might be helped by reducing how far your multicast messaging goes in your network.
- Reduce multicast latency by deactivating batching. By default, GemFire uses batching for operations when the region's scope is distributed-no-ack. Set the `disableBatching` property to true on the application or when starting a cache server process through the `gfsh` command line:

  ```
  gfsh>start server --name=server_name --J=-Dp2p.disableBatching=true
  ```

Run-time Considerations for Multicast

When you use multicast for messaging and data distribution, you need to understand how the health monitoring setting works and how to control memory use.

Multicast Health Monitor

The VMware GemFire management and monitoring system is supplemented by a `maxRetransmissionRatio` health monitoring setting for cluster members. This ratio is the number of retransmission requests received divided by the number of multicast datagrams written. If the ratio is at 1.0, the member is retransmitting as many packets as it originally sent. Retransmissions are point-to-point, and many processes may request retransmission, so this number can get quite high if problems occur. The default value for `maxRetransmissionRatio` is 0.2.

For example, consider a cluster with one producer and two consumers of cache events using multicast to transmit cache updates. The new member is added, which is running on a machine without multicast enabled. As a result, there is a retransmission request for every cache update, and the `maxRetransmissionRatio` changes to 1.0.

Controlling Memory Use on VMware GemFire Hosts with Multicast

Running out of memory can impede a member's performance and eventually lead to severe errors.

When data is distributed over multicast, VMware GemFire incurs a fixed overhead of memory reserved for transmission buffers. A specified amount of memory is reserved for each distributed region. These producer-side buffers are used only when a receiver is not getting enough CPU to read from its own receiving buffer as quickly as the producer is sending. In this case, the receiver complains of lost data. The producer then retrieves the data, if it still exists in its buffer, and resends to the receiver.

Tuning the transmission buffers requires a careful balance. Larger buffers mean that more data remains available for retransmission, providing more protection in case of a problem. On the other hand, a larger amount of reserved memory means that less memory is available for caching.

You can adjust the transmission buffer size by resetting the `mcast-send-buffer-size` parameter in the `gemfire.properties` file:

```
mcast-send-buffer-size=45000
```

Note: The maximum buffer size is constrained only by the limits of your system. If you are not seeing problems that could be related to lack of memory then do not change the default, since it provides greater protection in case of network problems.

Troubleshooting the Multicast Tuning Process

Several problems may arise during the initial testing and tuning process for multicasting.

Some or All Members Cannot Communicate

If your applications and cache servers cannot talk to each other, even though they are configured correctly, you may not have multicast connectivity on your network. It's common to have unicast connectivity, but not multicast connectivity. See your network administrator.

Multicast Is Slower Than Expected

Look for an Ethernet flow control limit. If you have mixed-speed networks that result in a multicast flooding problem, the Ethernet hardware may be trying to slow down the fast traffic.

Make sure your network hardware can deal with multicast traffic and route it efficiently. Some network hardware designed to handle multicast does not perform well enough to support a full-scale production system.

Multicast Fails Unexpectedly

If you find through testing that multicast fails above a round number, for example, it works up to 100 Mbps and fails at all rates over that, suspect that it is failing because it exceeds the network rate. This problem often arises at sites where one of the secondary LANs is slower than the main network

Maintaining Cache Consistency

Maintaining data consistency between caches in a distributed VMware GemFire system is vital for ensuring its functional integrity and preventing data loss.

General Guidelines

Before Restarting a Region with a Disk Store, Consider the State of the Entire Region

Note: If you revoke a member's disk store, do not restart that member with its disk stores—in isolation—at a later time.

VMware GemFire stores information about your persisted data and prevents you from starting a member with a revoked disk store in the running system. But VMware GemFire cannot stop you from starting a revoked member in isolation, and running with its revoked data. This is an unlikely situation, but it is possible to do:

1. Members A and B are running, both storing Region data to disk.
2. Member A goes down.
3. Member B goes down.
4. At this point, Member B has the most recent disk data.
5. Member B is not usable. Perhaps its host machine is down or cut off temporarily.
6. To get the system up and running, you start Member A, and use the command line tool to revoke Member B's status as member with the most recent data. The system loads Member A's data and you run forward with that.

7. Member A is stopped.
8. At this point, both Member A and Member B have information in their disk files indicating they are the gold copy members.
9. If you start Member B, it will load its data from disk.
10. When you start Member A, the system will recognize the incompatible state and report an exception, but by this point, you have good data in both files, with no way to combine them.

Understand Cache Transactions

Understanding the operation of VMware GemFire transactions can help you minimize situations where the cache could get out of sync.

Transactions do not work in distributed regions with global scope.

Transactions provide consistency within one cache, but the distribution of results to other members is not as consistent.

Multiple transactions in a cache can create inconsistencies because of read committed isolation. Since multiple threads cannot participate in a transaction, most applications will be running multiple transactions.

An in-place change to directly alter a key's value without doing a put can result in cache inconsistencies. With transactions, it creates additional difficulties because it breaks read committed isolation. If at all possible, use copy-on-read instead.

In distributed-no-ack scope, two conflicting transactions in different members can commit simultaneously, overwriting each other as the changes are distributed.

If a cache writer exists during a transaction, then each transaction write operation triggers a cache writer's related call. Regardless of the region's scope, a transaction commit can invoke a cache writer only in the local cache and not in the remote caches.

A region in a cache with transactions may not stay in sync with a region of the same name in another cache without transactions.

Two applications running the same sequence of operations in their transactions may get different results. This could occur because operations happening outside a transaction in one of the members can overwrite the transaction, even in the process of committing. This could also occur if the results of a large transaction exceed the machine's memory or the capacity of VMware GemFire. Those limits can vary by machine, so the two members may not be in sync.

Guidelines for Multi-Site Deployments

Optimize socket-buffer-size

In a multi-site installation using gateways, if the link between sites is not tuned for optimum throughput, it could cause messages to back up in the cache queues. If a queue overflows because of inadequate buffer sizes, it will become out of sync with the sender and the receiver will be unaware of the condition. You can configure the send-receive buffer sizes of the TCP/IP connections used for data transmissions by changing the socket-buffer-size attribute of the gateway-sender and gateway-receiver elements in the `cache.xml` file. Set the buffer size by determining the link bandwidth and then using ping to measure the round-trip time.

When optimizing socket-buffer sizes, use the same value for both gateway senders and gateway receivers.

Prevent Primary and Secondary Gateway Senders from Going Offline

In a multi-site installation, if the primary gateway server goes offline, a secondary gateway sender must take over primary responsibilities as the failover system. The existing secondary gateway sender detects that the primary gateway sender has gone offline, and a secondary one becomes the new primary. Because the queue is distributed, its contents are available to all gateway senders. So, when a secondary gateway sender becomes primary, it is able to start processing the queue where the previous primary left off with no loss of data.

If both the primary gateway sender and all its secondary senders go offline and messages are in their queues, data loss could occur, because there is no failover system.

Verify That isOriginRemote Is Set to False

The isOriginRemote flag for a server or a multi-site gateway is set to false by default, which ensures that updates are distributed to other members. Setting its value to true in the server or the receiving gateway member applies updates to that member only, so updates are not distributed to peer members.

Logging

Comprehensive logging messages help you confirm system configuration and debug problems in configuration and code.

- **How VMware GemFire Logging Works**

  VMware GemFire uses Apache Log4j 2 as the basis for its logging system.

- **Understanding Log Messages and Their Categories**

  System logging messages typically pertain to startup; logging management; connection and system membership; distribution; or cache, region, and entry management.

- **Naming, Searching, and Creating Log Files**

  The best way to manage and understand the logs is to have each member log to its own files.

- **Set Up Logging**

  You configure logging in a member's `gemfire.properties` or at startup with `gfsh`.

- **Advanced Users—Configuring Log4j 2 for VMware GemFire**

  Basic VMware GemFire logging configuration is configured via the gemfire.properties file. This topic is intended for advanced users who need increased control over logging due to integration with third-party libraries.

How VMware GemFire Logging Works

VMware GemFire uses Apache Log4j 2 as the basis for its logging system.

VMware GemFire uses Apache Log4j 2 API and Core libraries as the basis for its logging system. Log4j 2 API is a popular and powerful front-end logging API used by all the VMware GemFire classes to generate log statements. Log4j 2 Core is a backend implementation for logging; you can route any of the front-end logging API libraries to log to this backend. VMware GemFire uses the Core backend to run two custom Log4j 2 Appenders: **AlertAppender** and **LogWriterAppender**.

VMware GemFire has been tested with Log4j 2.17.1. VMware GemFire requires the `log4j-api-2.17.1.jar` and `log4j-core-2.17.1.jar` JAR files to be in the classpath. Both of these JARs are distributed in the `<path-to-product>/lib` directory and included in the appropriate `*-dependencies.jar` convenience libraries.

AlertAppender is the component that generates VMware GemFire alerts that are then managed by the JMX Management and Monitoring system. See Notification Federation for more details.

LogWriterAppender is the component that is configured by all the `log-*` VMware GemFire properties such as `log-file`, `log-file-size-limit` and `log-disk-space-limit`.

Both of these appenders are created and controlled programmatically. You configure their behavior with the `log-*` VMware GemFire properties and the alert level that is configured within the JMX Management & Monitoring system. These appenders do not currently support configuration within a `log4j2.xml` config file.

Advanced users may wish to define their own `log4j2.xml`. See Advanced Users—Configuring Log4j 2 for VMware GemFire for more details.

Understanding Log Messages and Their Categories

System logging messages typically pertain to startup; logging management; connection and system membership; distribution; or cache, region, and entry management.

- **Startup information**. Describe the Java version, the VMware GemFire native version, the host system, current working directory, and environment settings. These messages contain all information about the system and configuration the process is running with.
- **Logging management**. Pertain to the maintenance of the log files themselves. This information is always in the main log file (see the discussion at Log File Name).
- **Connections and system membership**. Report on the arrival and departure of cluster members (including the current member) and any information related to connection activities or failures. This includes information on communication between tiers in a hierarchical cache.
- **Distribution**. Report on the distribution of data between system members. These messages include information about region configuration, entry creation and modification, and region and entry invalidation and destruction.
- **Cache, region, and entry management**. Cache initialization, listener activity, locking and unlocking, region initialization, and entry updates.

Structure of a Log Message

Every logged message contains:

- The message header within square brackets:
    1. The message level
    2. The time the message was logged

3. The ID of the connection and thread that logged the message, which might be the main program or a system management process

- The message itself, which can be a string and/or an exception with the exception stack trace

```
[config 2005/11/08 15:46:08.710 PST PushConsumer main nid=0x1]
Cache initialized using "file:/Samples/quickstart/xml/PushConsumer.xml".
```

Log File Name

Specify your VMware GemFire system member's main log in the gemfire property `log-file` setting.

VMware GemFire uses this name for the most recent log file, actively in use if the member is running, or used for the last run. VMware GemFire creates the main log file when the application starts.

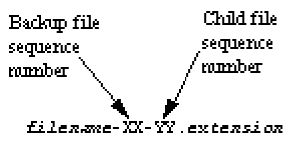
By default, the main log contains the entire log for the member session. If you specify a `log-file-size-limit`, VMware GemFire splits the logging into these files:

- **The main, current log**. Holding current logging entries. Named with the string you specified in `log-file`.
- **Child logs**. Holding older logging entries. These are created by renaming the main, current log when it reaches the size limit.
- **A metadata log file, with `meta-` prefixed to the name**. Used to track of startup, shutdown, child log management, and other logging management operations

The current log is renamed, or rolled, to the next available child log when the specified size limit is reached.

When your application connects with logging enabled, it creates the main log file and, if required, the `meta-` log file. If the main log file is present when the member starts up, it is renamed to the next available child log to make way for new logging.

Your current, main log file always has the name you specified in `log-file`. The old log files and child log files have names derived from the main log file name. These are the pieces of a renamed log or child log file name where `filename.extension` is the `log-file` specification



If child logs are not used, the child file sequence number is a constant 00 (two zeros).

For locators, the log file name is fixed. For the standalone locator started in `gfsh`, it is always named `<locator_name>.log` where the locator_name corresponds to the name specified at locator startup. For the locator that runs colocated inside another member, the log file is the member's log file.

For applications and the servers, your log file specification can be relative or absolute. If no file is specified, the defaults are standard output for applications and `<server_name>.log` for servers started with gfsh and `cacheserver.log` for servers started with the older cacheserver script.

To figure out the member's most recent activities, look at the `meta-` log file or, if no meta file exists, the main log file.

How the System Renames Logs

The log file that you specify is the base name used for all logging and logging archives. If a log file with the specified name already exists at startup, the cluster automatically renames it before creating the current log file. This is a typical directory listing after a few runs with `log-file=system.log`:

```
bash-2.05$ ls -tlra system*
-rw-rw-r-- 1 jpearson users 11106 Nov 3 11:07 system-01-00.log
-rw-rw-r-- 1 jpearson users 11308 Nov 3 11:08 system-02-00.log
-rw-rw-r-- 1 jpearson users 11308 Nov 3 11:09 system.log
bash-2.05$
```

The first run created `system.log` with a timestamp of Nov 3 11:07. The second run renamed that file to `system-01-00.log` and created a new `system.log` with a timestamp of Nov 3 11:08. The third run renamed that file to `system-02-00.log` and created the file named `system.log` in this listing.

When the cluster renames the log file, it assigns the next available number to the new file, as XX of `filename-XX-YY.extension`. This next available number depends on existing old log files and also on any old statistics archives. The system assigns the next number that is higher than any in use for statistics or logging. This keeps current log files and statistics archives paired up regardless of the state of the older files in the directory. Thus, if an application is archiving statistics and logging to `system.log` and `statArchive.gfs`, and it runs in a Unix directory with these files:

```
bash-2.05$ ls -tlr stat* system*
-rw-rw-r-- 1 jpearson users 56143 Nov 3 11:07 statArchive-01-00.gfs
-rw-rw-r-- 1 jpearson users 56556 Nov 3 11:08 statArchive-02-00.gfs
-rw-rw-r-- 1 jpearson users 56965 Nov 3 11:09 statArchive-03-00.gfs
-rw-rw-r-- 1 jpearson users 11308 Nov 3 11:27 system-01-00.log
-rw-rw-r-- 1 jpearson users 59650 Nov 3 11:34 statArchive.gfs
-rw-rw-r-- 1 jpearson users 18178 Nov 3 11:34 system.log
```

the directory contents after the run look like this (changed files in **bold**):

```
bash-2.05$ ls -ltr stat* system*
-rw-rw-r-- 1 jpearson users 56143 Nov 3 11:07 statArchive-01-00.gfs
-rw-rw-r-- 1 jpearson users 56556 Nov 3 11:08 statArchive-02-00.gfs
-rw-rw-r-- 1 jpearson users 56965 Nov 3 11:09 statArchive-03-00.gfs
-rw-rw-r-- 1 jpearson users 11308 Nov 3 11:27 system-01-00.log
-rw-rw-r-- 1 jpearson users 59650 Nov 3 11:34 statArchive-04-00.gfs
-rw-rw-r-- 1 jpearson users 18178 Nov 3 11:34 system-04-00.log
-rw-rw-r-- 1 jpearson users 55774 Nov 4 10:08 statArchive.gfs
-rw-rw-r-- 1 jpearson users 17681 Nov 4 10:08 system.log
```

The statistics and the log file are renamed using the next integer that is available to both, so the log file sequence jumps past the gap in this case.

Log Level

The higher the log level, the more important and urgent the message. If you are having problems with your system, a first-level approach is to lower the log-level (thus sending more of the detailed messages to the log file) and recreate the problem. The additional log messages often help uncover the source.

These are the levels, in descending order, with sample output:

- **severe (highest level)**. This level indicates a serious failure. In general, severe messages describe events that are of considerable importance that will prevent normal program execution. You will likely need to shut down or restart at least part of your system to correct the situation.

  This severe error was produced by configuring a system member to connect to a non-existent locator:

  ```
  [severe 2005/10/24 11:21:02.908 PDT nameFromGemfireProperties
  DownHandler (FD_SOCK) nid=0xf] GossipClient.getInfo():
  exception connecting to host localhost:30303:
  java.net.ConnectException: Connection refused
  ```

- **error**. This level indicates that something is wrong in your system. You should be able to continue running, but the operation noted in the error message failed.

  This error was produced by throwing a `Throwable` from a `CacheListener`. While dispatching events to a customer-implemented cache listener, VMware GemFire catches any `Throwable` thrown by the listener and logs it as an error. The text shown here is followed by the output from the `Throwable` itself.

  ```
  [error 2007/09/05 11:45:30.542 PDT gemfire1_newton_18222
  <vm_2_thr_5_client1_newton_18222-0x472e> nid=0x6d443bb0]
  Exception occurred in CacheListener
  ```

- **warning**. This level indicates a potential problem. In general, warning messages describe events that are of interest to end users or system managers, or that indicate potential problems in the program or system.

  This message was obtained by starting a client with a Pool configured with queueing enabled when there was no server running to create the client's queue:

  ```
  [warning 2008/06/09 13:09:28.163 PDT <queueTimer-client> tid=0xe]
  QueueManager - Could not create a queue. No queue servers available
  ```

  This message was obtained by trying to get an entry in a client region while there was no server running to respond to the client request:

  ```
  [warning 2008/06/09 13:12:31.833 PDT <main> tid=0x1] Unable to create a
  connection in the allowed time
  org.apache.geode.cache.client.NoAvailableServersException
      at org.apache.geode.cache.client.internal.pooling.ConnectionManagerImpl.
  borrowConnection(ConnectionManagerImpl.java:166)
  . . .
  org.apache.geode.internal.cache.LocalRegion.get(LocalRegion.java:1122
  )
  ```

- **info**. This is for informational messages, typically geared to end users and system administrators.

  This is a typical info message created at system member startup. This indicates that no other `DistributionManager`s are running in the cluster, which means no other system members are running:

  ```
  [info 2005/10/24 11:51:35.963 PDT CacheRunner main nid=0x1]
  DistributionManager straw(7368):41714 started on 192.0.2.0[10333]
  with id straw(7368):41714 (along with 0 other DMs)
  ```

  When another system member joins the cluster, these info messages are output by the members that are already running:

  ```
  [info 2005/10/24 11:52:03.934 PDT CacheRunner P2P message reader for
  straw(7369):41718 nid=0x21] Member straw(7369):41718 has joined the
  distributed cache.
  ```

  When another member leaves because of an interrupt or through normal program termination:

  ```
  [info 2005/10/24 11:52:05.128 PDT CacheRunner P2P message reader for
  straw(7369):41718 nid=0x21] Member straw(7369):41718 has left the
  distributed cache.
  ```

  And when another member is unexpectedly terminated:

  ```
  [info 2005/10/24 13:08:41.389 PDT CacheRunner DM-Puller nid=0x1b] Member
  straw(7685):41993 has unexpectedly left the distributed cache.
  ```

- **config**. This is the default setting for logging. This level provides static configuration messages that are often used to debug problems associated with particular configurations.

  You can use this config message to verify your startup configuration:

  ```
  [config 2008/08/08 14:28:19.862 PDT CacheRunner <main> tid=0x1] Startup Configu
  ration:
  ack-severe-alert-threshold="0"
  ack-wait-threshold="15"
  archive-disk-space-limit="0"
  archive-file-size-limit="0"
  async-distribution-timeout="0"
  async-max-queue-size="8"
  async-queue-timeout="60000"
  bind-address=""
  cache-xml-file="cache.xml"
  conflate-events="server"
  conserve-sockets="false"
    ...
  socket-buffer-size="32768"
  socket-lease-time="60000"
  ssl-ciphers="any"
  ssl-enabled="false"
  ssl-protocols="any"
  ssl-require-authentication="true"
  start-locator=""
  ```

```
statistic-archive-file=""
statistic-sample-rate="1000"
statistic-sampling-enabled="false"
tcp-port="0"
udp-fragment-size="60000"
udp-recv-buffer-size="1048576"
udp-send-buffer-size="65535"
```

- **fine**. This level provides tracing information that is generally of interest to developers. It is used for the lowest volume, most important, tracing messages.

  Note: Generally, you should only use this level if instructed to do so by technical support. At this logging level, you will see a lot of noise that might not indicate a problem in your application. This level creates very verbose logs that may require significantly more disk space than the higher levels.

  ```
  [fine 2011/06/21 11:27:24.689 PDT <locatoragent_ds_w1-gst-dev04_2104> tid=0xe]
  SSL Configuration:
      ssl-enabled = false
  ```

- **finer, finest, and all**. These levels exist for internal use only. They produce a large amount of data and so consume large amounts of disk space and system resources. **Note:** Do not use these settings unless asked to do so by technical support.

Note: VMware GemFire no longer supports setting system properties for VERBOSE logging. To enable VERBOSE logging, see Advanced Users—Configuring Log4j 2 for VMware GemFire

Naming, Searching, and Creating Log Files

The best way to manage and understand the logs is to have each member log to its own files.

Log File Naming Recommendation

For members running on the same machine, you can have them log to their own files by starting them in different working directories and using the same, relative `log-file` specification. For example, you could set this in `<commonDirectoryPath>/gemfire.properties`:

```
log-file=./log/member.log
```

then start each member in a different directory with this command, which points to the common properties file:

```
java -DgemfirePropertyFile=<commonDirectoryPath>/gemfire.properties
```

This way, each member has its own log files under its own working directory.

Searching the Log Files

For the clearest picture, merge the log files, with the `gfsh export logs` command:

```
gfsh> export logs --dir=myDir --dir=myDir --merge-log=true
```

Search for lines that begin with these strings:

- [warning
- [error
- [severe

Creating Your Own Log Messages

In addition to the system logs, you can add your own application logs from your Java code. For information on adding custom logging to your applications, see the online Java documentation for the `org.apache.geode.LogWriter` interface. Both system and application logging is output and stored according to your logging configuration settings.

Set Up Logging

You configure logging in a member's `gemfire.properties` or at startup with `gfsh`.

Before you begin, make sure you understand Basic Configuration and Programming.

1. Run a time synchronization service such as NTP on all VMware GemFire host machines. This is the only way to produce logs that are useful for troubleshooting. Synchronized time stamps ensure that log messages from different hosts can be merged to accurately reproduce a chronological history of a distributed run.
2. Use a sniffer to monitor your logs Look for new or unexpected warnings, errors, or severe messages. The logs output by your system have their own characteristics, indicative of your system configuration and of the particular behavior of your applications, so you must become familiar with your applications' logs to use them effectively.
3. Configure member logging in each member's `gemfire.properties` as needed:

   ```
   # Default gemfire.properties log file settings
   log-level=config
   log-file=
   log-file-size-limit=0
   log-disk-space-limit=0
   ```

   Note: You can also specify logging parameters when you start up members (either locators or servers) using the `gfsh` command-line utility. In addition, you can modify log file properties and log-level settings while a member is already running by using the alter runtime command.

   1. Set `log-level`. Log levels are not case-sensitive. Options are:
      - `none` or `off`
      - `severe` or `fatal` (log messages with the highest level of importance)
      - `error`
      - `warning` or `warn`
      - `info` or `config`
      - `fine` or `debug`

        - `finer` or `trace`
        - `finest` or `trace`
        - `all`

        The more detailed levels (lower in this list) also include messages from all levels above them. For example, a `warning` log level shows `warning`, `error`, and `severe` messages. For general troubleshooting, we recommend setting the log level at `config` or higher on this list. The most detailed, lower levels in this list (`all`, `finest`, `finer`, `fine`) can quickly consume disk space and negatively impact system performance. Use these more detailed levels only when necessary.

    2. Specify the log file name in `log-file`. This can be relative or absolute. If this property is not specified, the defaults are:

        - Standard output for applications
        - For servers, the default log file location is:

            ```
            working-directory/server-name.log
            ```

            By default, when starting a server through `gfsh`, the *working -directory* corresponds to the directory (named after itself) that the cache server creates upon startup. Alternatively, you can specify a different working directory path when you start the cache server. The *server-name* corresponds to the name of the cache server provided upon startup.

        - For a standalone locator, the default log file location is:

            ```
            working-directory/locator-name.log
            ```

            By default, when starting a locator through `gfsh`, the *working -directory* corresponds to the directory (named after itself) created when the locator starts up. Alternatively, you can specify a different working directory path when you start a locator. The *locator-name* corresponds to the name of the locator provided upon startup. If you are using a colocated or embedded locator, the locator logs will be part of the member's log file.

        For the easiest logs examination and troubleshooting, send your logs to files instead of standard out. **Note:** Make sure each member logs to its own files. This makes the logs easier to decipher.

    3. Set the maximum size of a single log file in `log-file-size-limit`. If not set, the single, main log file is used. If set, the metadata file, the main log, and rolled child logs are used.

    4. Set the maximum size of all log files in `log-disk-space-limit`. If non-zero, this limits the combined size of all inactive log files, deleting oldest files first to stay under the limit. A zero setting indicates no limit.

4. If you are using the `gfsh` command-line interface, `gfsh` can create its own log file in the directory where you run the `gfsh` or `gfsh.bat` script. By default, gfsh does not generate log files for itself. To enable gfsh logs, set the following system property to the desired log level before starting gfsh:

```
export JAVA_ARGS=-Dgfsh.log-level=[severe|warning|info|config|fine|finer|fines
t]
```

gfsh log files are named `gfsh-0_0.log`.

Advanced Users—Configuring Log4j 2 for VMware GemFire

Basic VMware GemFire logging configuration is configured via the `gemfire.properties` file. This topic is intended for advanced users who need increased control over logging due to integration with third-party libraries.

An example `log4j2.xml` can be located within the product distribution at `$GEODE/config/log4j2.xml`.

To specify your own `log4j2.xml` configuration file (or anything else supported by Log4j 2 such as .json or .yaml), use the following flag when starting up your JVM or VMware GemFire member:

```
-Dlog4j.configurationFile=<location-of-your-file>
```

If the Java system property `log4j.configurationFile` is specified, then Log4j will not use the `log4j2.xml` included in `geode-log4j-<version>.jar`.

Using Different Front-End Logging APIs to Log to Log4j2

You can also configure Log4j 2 to work with various popular and commonly used logging APIs. To obtain and configure the most popular front-end logging APIs to log to Log4j 2, see the instructions on the Apache Log4j 2 web site at http://logging.apache.org/log4j/2.x/.

For example, if you are using:

- **Commons Logging**, download “Commons Logging Bridge” (`log4j-jcl-2.17.1.jar`)
- **SLF4J**, download “SLFJ4 Binding” (`log4j-slf4j-impl-2.17.1.jar`)
- **java.util.logging**, download the “JUL adapter” (`log4j-jul-2.17.1.jar`)

See http://logging.apache.org/log4j/2.x/faq.html for more examples.

All three of the above JAR files are in the full distribution of Log4J 2.17.1 which can be downloaded at http://logging.apache.org/log4j/2.x/download.html. Download the appropriate bridge, adapter, or binding JARs to ensure that VMware GemFire logging is integrated with every logging API used in various third-party libraries or in your own applications.

Note: VMware GemFire has been tested with Log4j 2.17.1. As newer versions of Log4j 2 come out, you can find 2.17.1 under Previous Releases on that page.

Customizing Your Own log4j2.xml File

Advanced users may want to move away entirely from setting `log-*` gemfire properties and instead specify their own `log4j2.xml` using `-Dlog4j.configurationFile`.

Custom Log4j 2 configuration in VMware GemFire comes with some caveats and notes:

- Do not use `"monitorInterval="` in your `log4j2.xml` file, because doing so can have significant performance impact. This setting instructs Log4j 2 to monitor the log4j2.xml config file at runtime and automatically reload and reconfigure if the file changes.

- VMware GemFire’s default `log4j2.xml` specifies status=“FATAL” because Log4j 2’s StatusLogger generates warnings to standard out at ERROR level anytime VMware GemFire stops its AlertAppender or LogWriterAppender. VMware GemFire uses a lot of concurrent threads that are executing code with log statements; these threads may be logging while the VMware GemFire appenders are being stopped.
- VMware GemFire’s default `log4j2.xml` specifies `shutdownHook="disable"` because VMware GemFire has a shutdown hook which disconnects the DistributedSystem and closes the Cache, which is executing the code that performs logging. If the Log4J2 shutdown hook stops logging before VMware GemFire completes its shutdown, Log4j 2 will attempt to start back up. This restart in turn attempts to register another Log4j 2 shutdown hook which fails resulting in a FATAL level message logged by Log4j 2.
- The GEODE_VERBOSE marker (Log4J2 Marker are discussed on http://logging.apache.org/log4j/2.x/manual/markers.html) can be used to enable additional verbose log statements at TRACE level. Many log statements are enabled simply by enabling DEBUG or TRACE. However, even more log statements can be further enabled by using MarkerFilter to accept GEODE_VERBOSE. The default VMware GemFire `log4j2.xml` disables GEODE_VERBOSE with this line:

  ```
  <MarkerFilter marker="GEODE_VERBOSE" onMatch="DENY" onMismatch="NEUTRAL"/>
  ```

  You can enable the GEODE_VERBOSE log statements by changing `onMatch="DENY"` to `onMatch="ACCEPT"`. Typically, it’s more useful to simply enable DEBUG or TRACE on certain classes or packages instead of for the entire VMware GemFire product. However, this setting can be used for internal debugging purposes if all other debugging methods fail.

Statistics

Every application and server in a distributed system can access statistical data about VMware GemFire operations. You can configure the gathering of statistics by using the `alter runtime` command of `gfsh` or in the `gemfire.properties` file to facilitate system analysis and troubleshooting.

- **How Statistics Work**

  Each application or cache server that joins the distributed system can collect and archive statistical data for analyzing system performance.
- **Transient Region and Entry Statistics**

  For replicated, distributed, and local regions, VMware GemFire provides a standard set of statistics for the region and its entries.
- **Application-Defined and Custom Statistics**

  VMware GemFire includes interfaces for defining and maintaining your own statistics.
- **Configuring and Using Statistics**

  You configure statistics and statistics archiving in several different ways.
- **Viewing Archived Statistics**

How Statistics Work

Each application or cache server that joins the cluster can collect and archive statistical data for analyzing system performance.

VMware GemFire statistics can be enabled for a cluster, for an application, for a server, or for a region. Statistics gathered for a cluster, an application, or a cache server are saved to a file and can be archived, whereas region statistics are transient and accessible only through the API.

Set the configuration attributes that control cluster, application, or cache statistics collection in `gfsh` or in the `gemfire.properties` configuration file. You can also collect your own application defined statistics.

When Java applications and servers join a cluster, they can be configured via the cluster configuration service to enable statistics sampling and whether to archive the statistics that are gathered.

Note: VMware GemFire statistics use the Java `System.nanoTimer` for nanosecond timing. This method provides nanosecond precision, but not necessarily nanosecond accuracy. For more information, see the online Java documentation for `System.nanoTimer` for the JRE you are using with VMware GemFire.

Statistics sampling provides valuable information for ongoing system tuning and troubleshooting. Sampling statistics at the default sample rate does not impact overall cluster performance. We recommend enabling statistics sampling in production environments.

Transient Region and Entry Statistics

For replicated, distributed, and local regions, VMware GemFire provides a standard set of statistics for the region and its entries.

VMware GemFire gathers these statistics when the `--enable-statistics` parameter of the `create region` command of `gfsh` is set to true or in cache.xml the region attribute `statistics-enabled` is set to true.

Note: Unlike other VMware GemFire statistics, these region and entry statistics are not archived and cannot be charted.

Note: Enabling these statistics requires extra memory per entry. See Memory Requirements for Cached Data.

These are the transient statistics gathered for all but partitioned regions:

- **Hit and miss counts**. For the entry, the hit count is the number of times the cached entry was accessed through the `Region.get` method and the miss count is the number of times these hits did not find a valid value. For the region these counts are the totals for all entries in the region. The API provides `get` methods for the hit and miss counts, a convenience method that returns the hit-to-miss ratio, and a method for zeroing the counts.
- **Last accessed time**. For the entry, this is the last time a valid value was retrieved from the locally cached entry. For the region, this is the most recent “last accessed time” for all entries contained in the region. This statistic is used for idle timeout expiration activities.
- **Last modified time**. For the entry, this is the last time the entry value was updated (directly or through distribution) due to a load, create, or put operation. For the region, this

is the most recent “last modified time” for all entries contained in the region. This statistic is used for time to live and idle timeout expiration activities.

The hit and miss counts collected in these statistics can be useful for fine-tuning your system’s caches. If you have a region’s entry expiration enabled, for example, and see a high ratio of misses to hits on the entries, you might choose to increase the expiration times.

Retrieve region and entry statistics through the `getStatistics` methods of the `Region` and `Region.Entry` objects.

Application-Defined and Custom Statistics

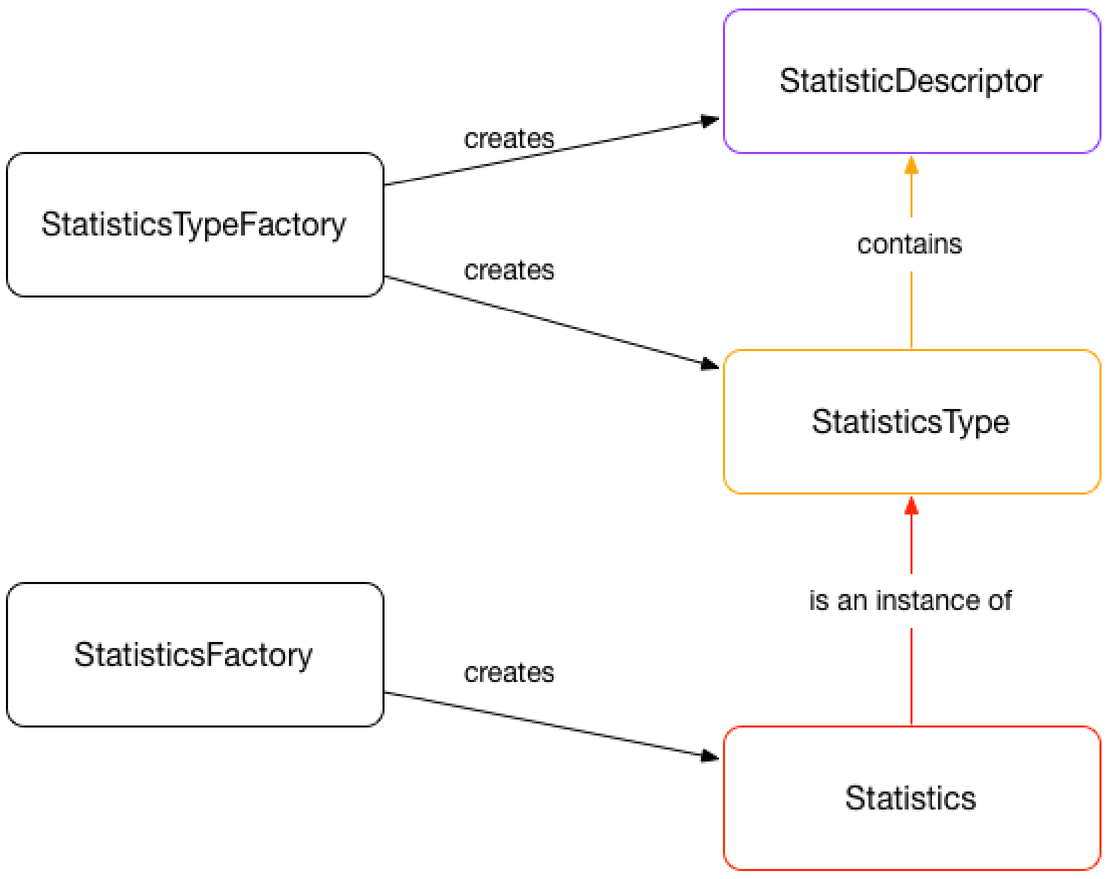
VMware GemFire includes interfaces for defining and maintaining your own statistics.

The VMware GemFire package, `org.apache.geode`, includes the following interfaces for defining and maintaining your own statistics:

- **StatisticDescriptor**. Describes an individual statistic. Each statistic has a name and information on the statistic it holds, such as its class type (long, int, etc.) and whether it is a counter that always increments, or a gauge that can vary in any manner.
- **StatisticsType**. Logical type that holds a list of `StatisticDescriptors` and provides access methods to them. The `StatisticDescriptors` contained by a `StatisticsType` are each assigned a unique ID within the list. `StatisticsType` is used to create a `Statistics` instance.
- **Statistics**. Instantiation of an existing `StatisticsType` object with methods for setting, incrementing, getting individual `StatisticDescriptor` values, and setting a callback which will recompute the statistic’s value at configured sampling intervals.
- **StatisticsFactory**. Creates instances of `Statistics`. You can also use it to create instances of `StatisticDescriptor` and `StatisticsType`, because it implements `StatisticsTypeFactory`. `DistributedSystem` is an instance of `StatisticsFactory`.
- **StatisticsTypeFactory**. Creates instances of `StatisticDescriptor` and `StatisticsType`.

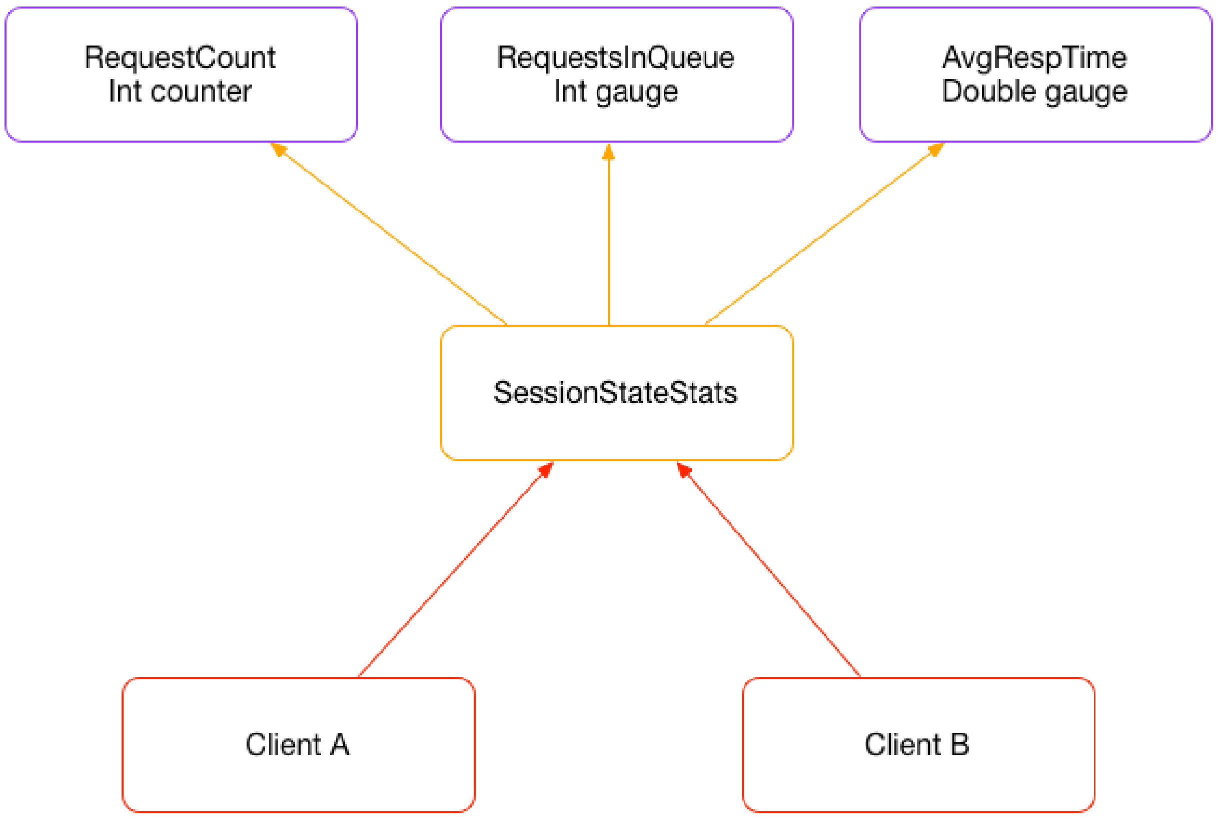
The statistics interfaces are instantiated using statistics factory methods that are included in the package. For coding examples, see the online Java API documentation for `StatisticsFactory` and `StatisticsTypeFactory`.

As an example, an application server might collect statistics on each client session in order to gauge whether client requests are being processed in a satisfactory manner. Long request queues or long server response times could prompt some capacity-management action such as starting additional application servers. To set this up, each session-state data point is identified and defined in a `StatisticDescriptor` instance. One instance might be a `RequestsInQueue` gauge, a non-negative integer that increments and decrements. Another could be a `RequestCount` counter, an integer that always increments. A list of these descriptors is used to instantiate a `SessionStateStats StatisticsType`. When a client connects, the application server uses the `StatisticsType` object to create a session-specific `Statistics` object. The server then uses the `Statistics` methods to modify and retrieve the client’s statistics. The figures below illustrate the relationships between the statistics interfaces and show the implementation of this use case.



The Statistics Interfaces

Each `StatisticDescriptor` contains one piece of statistical information. `StatisticalDesriptor` objects are collected into a `StatisticsType`. The `StatisticsType` is instantiated to create a `Statistics` object.



Statistics Implementation

The `StatisticDescriptor` objects shown here hold three pieces of statistical information about client session state. These are collected into a `SessionStateStats StatisticsType`. With this type, the server creates a `Statistics` object for each client that connects.

Configuring and Using Statistics

You can configure statistics and statistics archiving in several ways.

Configure Cluster or Server Statistics

In this procedure it is assumed that you understand Basic Configuration and Programming.

Execute the following commands to modify the cluster's configuration and enable cluster or server statistics.

```
gfsh>start locator --name=l1 --enable-cluster-configuration=true
gfsh>alter runtime --enable-statistics=true --statistic-archive-file=myStatisticsArchi
veFile.gfs
```

Note that an empty value of `statistic-archive-file` still calculates statistics, but they are not archived to a file.

You can also configure sample rate and the filename of your statistic archive files. See alter runtime for more command options.

Alternately, if you are not using the cluster configuration service, configure `gemfire.properties` for the statistics monitoring and archiving that you need:

1. Enable statistics gathering for the cluster. This is required for all other statistics activities:

   ```
   statistic-sampling-enabled=true
   statistic-archive-file=myStatisticsArchiveFile.gfs
   ```

   Note: Statistics sampling at the default sample rate (1000 milliseconds) does not impact system performance and is recommended in production environments for troubleshooting.

2. Change the statistics sample rate as needed. Example:

   ```
   statistic-sampling-enabled=true
   statistic-archive-file=myStatisticsArchiveFile.gfs
   statistic-sample-rate=2000
   ```

3. To archive the statistics to disk, enable that and set any file or disk space limits that you need. Example:

   ```
   statistic-sampling-enabled=true
   statistic-archive-file=myStatisticsArchiveFile.gfs
   archive-file-size-limit=100
   archive-disk-space-limit=1000
   ```

4. If you need time-based statistics, enable that. Time-based statistics require statistics sampling and archiving. This setting also enables Micrometer meters of type timer. Example:

   ```
   statistic-sampling-enabled=true
   statistic-archive-file=myStatisticsArchiveFile.gfs
   enable-time-statistics=true
   ```

If these statistics are on, you are able to access archived statistics through the `gfsh show metrics` command.

Configure Transient Region and Entry Statistics

Enable transient region and entry statistics gathering on the regions where you need them. This configuration is distinct from the enabling of cluster or server statistics.

gfsh example:

```
gfsh>create region --name=myRegion --type=REPLICATE --enable-statistics=true
```

cache.xml example:

```
<region name="myRegion" refid="REPLICATE">
<region-attributes statistics-enabled="true">
</region-attributes>
</region>
```

API example:

Note: Region and entry statistics are not archived and can be accessed only through the API. As needed, retrieve region and entry statistics through the `getStatistics` methods of the `Region` and `Region.Entry` objects. Example:

```
out.println("Current Region:\n\t" + this.currRegion.getName());
RegionAttributes attrs = this.currRegion.getAttributes();
  if (attrs.getStatisticsEnabled()) {
      CacheStatistics stats = this.currRegion.getStatistics();
      out.println("Stats:\n\tHitCount is " + stats.getHitCount() +
          "\n\tMissCount is " + stats.getMissCount() +
          "\n\tLastAccessedTime is " + stats.getLastAccessedTime() +
          "\n\tLastModifiedTime is " + stats.getLastModifiedTime());
  }
```

Configure Custom Statistics

Create and manage any custom statistics that you need through `cache.xml` and the API.

cache/cluster.xml example:

```
// Create custom statistics
<?xml version="1.0" encoding="UTF-8"?>
  <!DOCTYPE statistics PUBLIC
    "-//Example Systems, Inc.//Example Statistics Type//EN"
    "http://www.example.com/dtd/statisticsType.dtd">
  <statistics>
    <type name="StatSampler">
      <description>Stats on the statistic sampler.</description>
      <stat name="sampleCount" storage="int" counter="true">
        <description>Total number of samples taken by this sampler.</description>
        <unit>samples</unit>
      </stat>
      <stat name="sampleTime" storage="long" counter="true">
        <description>Total amount of time spent taking samples.</description>
        <unit>milliseconds</unit>
      </stat>
    </type>
</statistics>
```

API example:

```
// Update custom stats through the API
this.samplerStats.incInt(this.sampleCountId, 1);
this.samplerStats.incLong(this.sampleTimeId, nanosSpentWorking / 1000000);
```

Controlling the Size of Archive Files

You can specify limits on the archive files for statistics using the gfsh `alter runtime` command. These are the areas of control:

- **Archive File Growth Rate**.
  - The `--statistic-sample-rate` parameter controls how often samples are taken, which affects the speed at which the archive file grows.
  - The `--statistic-archive-file` parameter controls whether the statistics files are compressed. If you give the file name a `.gz` suffix, it is compressed, thereby taking up less disk space.

- **Maximum Size of a Single Archive File**. If the value of the `--archive-file-size-limit` is greater than zero, a new archive is started when the size of the current archive exceeds the limit. Only one archive can be active at a time. **Note:** If you modify the value of `--archive-file-size-limit` while the cluster is running, the new value does not take effect until the current archive becomes inactive (that is, when a new archive is started).
- **Maximum Size of All Archive Files**. The `--archive-disk-space-limit` parameter controls the maximum size of all inactive archive files combined. By default, the limit is set to 0, meaning that archive space is unlimited. Whenever an archive becomes inactive or when the archive file is renamed, the combined size of the inactive files is calculated. If the size exceeds the `--archive-disk-space-limit`, the inactive archive with the oldest modification time is deleted. This continues until the combined size is less than the limit. If `--archive-disk-space-limit` is less than or equal to `--archive-file-size-limit`, when the active archive is made inactive due to its size, it is immediately deleted.

Note: If you modify the value of `--archive-disk-space-limit` while the cluster is running, the new value does not take effect until the current archive becomes inactive.

Viewing Archived Statistics

When sampling and archiving are enabled, you can examine archived historical data to help diagnose performance problems. Study statistics in archive files by using the gfsh `show metrics` command or through the Visual Statistics Display tool (VSD) provided with GemFire. The VSD tool reads the sampled statistics and produces graphical displays for analysis.

Troubleshooting and System Recovery

This section provides strategies for handling common errors and failure situations.

- **Producing Artifacts for Troubleshooting**

  There are several types of files that are critical for troubleshooting.
- **Diagnosing System Problems**

  This section provides possible causes and suggested responses for system problems.
- **System Failure and Recovery**

  This section describes alerts for and appropriate responses to various kinds of system failures. It also helps you plan a strategy for data recovery.
- **Handling Forced Cache Disconnection Using Autoreconnect**

  A VMware GemFire member may be forcibly disconnected from a cluster if the member is unresponsive for a period of time, or if a network partition separates one or more members into a group that is too small to act as the cluster.
- **Recovering from Application and Cache Server Crashes**

  When the application or cache server crashes, its local cache is lost, and any resources it owned (for example, distributed locks) are released. The member must recreate its local cache upon recovery.
- **Recovering from Machine Crashes**

When a machine crashes because of a shutdown, power loss, hardware failure, or operating system failure, all of its applications and cache servers and their local caches are lost.

- **Recovering from ConflictingPersistentDataExceptions**

  A `ConflictingPersistentDataException` while starting up persistent members indicates that you have multiple copies of some persistent data, and VMware GemFire cannot determine which copy to use.

- **Preventing and Recovering from Disk Full Errors**

  It is important to monitor the disk usage of VMware GemFire members. If a member lacks sufficient disk space for a disk store, the member attempts to shut down the disk store and its associated cache, and logs an error message. A shutdown due to a member running out of disk space can cause loss of data, data file corruption, log file corruption and other error conditions that can negatively impact your applications.

- **Understanding and Recovering from Network Outages**

  The safest response to a network outage is to restart all the processes and bring up a fresh data set.

- **Log Messages and Solutions**

  This section provides strategies for responding to a variety of system log messages.

Producing Artifacts for Troubleshooting

There are several types of files that are critical for troubleshooting.

Geode logs and statistics are the two most important artifacts used in troubleshooting. In addition, they are required for Geode system health verification and performance analysis. For these reasons, logging and statistics should always be enabled, especially in production. Save the following files for troubleshooting purposes:

- Log files. Even at the default logging level, the log contains data that may be important. Save the whole log, not just the stack. For comparison, save log files from before, during, and after the problem occurred.
- Statistics archive files.
- Core files or stack traces.
- For Linux, you can use gdb to extract a stack from a core file.
- Crash dumps.
- For Windows, save the user mode dump files. Some locations to check for these files:
  - C:\ProgramData\Microsoft\Windows\WER\ReportArchive
  - C:\ProgramData\Microsoft\Windows\WER\ReportQueue
  - C:\Users*UserProfileName*\AppData\Local\Microsoft\Windows\WER\ReportArchive
  - C:\Users*UserProfileName*\AppData\Local\Microsoft\Windows\WER\ReportQueue

When a problem arises that involves more than one process, a network problem is the most likely cause. When you diagnose a problem, create a log file for each member of all the clusters involved. If you are running a client/server architecture, create log files for the clients.

Note: You must run a time synchronization service on all hosts for troubleshooting. Synchronized time stamps ensure that log messages on different hosts can be merged to accurately reproduce a chronological history of a distributed run.

For each process, complete these steps:

1. Make sure the host's clock is synchronized with the other hosts. Use a time synchronization tool such as Network Time Protocol (NTP).
2. Enable logging to a file instead of standard output by editing `gemfire.properties` to include this line:

   ```
   log-file=filename
   ```

3. Keep the log level at `config` to avoid filling up the disk while including configuration information. Add this line to `gemfire.properties`:

   ```
   log-level=config
   ```

   Note: Running with the log level at `fine` can impact system performance and fill up your disk.

4. Enable statistics gathering for the cluster either by modifying `gemfire.properties`:

   ```
   statistic-sampling-enabled=true
   statistic-archive-file=StatisticsArchiveFile.gfs
   ```

   or by using the `gfsh alter rutime` command:

   ```
   alter runtime --group=myMemberGroup --enable-statistics=true --statistic-archiv
   e-file=StatisticsArchiveFile.gfs
   ```

   Note: Collecting statistics at the default sample rate frequency of 1000 milliseconds does not incur performance overhead.

5. Run the application again.
6. Examine the log files. To get the clearest picture, merge the files. To find all the errors in the log file, search for lines that begin with these strings:

   ```
   [error
   [severe
   ```

   For details on merging log files, see the `--merge-log` argument for the export logscommand.

7. Export and analyze the stack traces on the member or member group where the application is running. Use the `gfsh export stack-traces command`. For example:

   ```
   gfsh> export stack-traces --file=ApplicationStackTrace.txt --member=member1
   ```

Diagnosing System Problems

This section provides possible causes and suggested responses for system problems.

Locator does not start

Invocation of a locator with gfsh fails with an error like this:

```
Starting a GemFire Locator in C:\devel\gfcache\locator\locator
The Locator process terminated unexpectedly with exit status 1. Please refer to the lo
g
        file in C:\devel\gfcache\locator\locator for full details.
Exception in thread "main" java.lang.RuntimeException: An IO error occurred while
        starting a Locator in C:\devel\gfcache\locator\locator on 192.0.2.0[10999]: Ne
twork is
        unreachable; port (10999) is not available on 192.0.2.0.
at
org.apache.geode.distributed.LocatorLauncher.start(LocatorLauncher.java:622)
```

```
at
org.apache.geode.distributed.LocatorLauncher.run(LocatorLauncher.java:513)
at
org.apache.geode.distributed.LocatorLauncher.main(LocatorLauncher.java:188)
Caused by: java.net.BindException: Network is unreachable; port (10999) is not availab
le on
        192.0.2.0.
at
org.apache.geode.distributed.AbstractLauncher.assertPortAvailable(AbstractLauncher.jav
a:136)
at
org.apache.geode.distributed.LocatorLauncher.start(LocatorLauncher.java:596)
...
```

This indicates a mismatch somewhere in the address, port pairs used for locator startup and configuration. The address you use for locator startup must match the address you list for the locator in the `gemfire.properties` locators specification. Every member of this cluster, including the locator itself, must have the complete locators specification in the `gemfire.properties`.

Response:

- Check that your locators specification includes the address you are using to start your locator.
- If you use a bind address, you must use numeric addresses for the locator specification. The bind address will not resolve to the machine's default address.
- If you are using a 64-bit Linux system, check whether your system is experiencing the leap second bug. See Java applications on 64-bit platforms hang or use 100% CPU for more information.

Application or cache server process does not start

If the process tries to start and then silently disappears, on Windows this indicates a memory problem.

Response:

- On a Windows host, decrease the maximum JVM heap size. This property is specified on the `gfsh` command line:

  ```
  gfsh>start server --name=server_name --max-heap=1024m
  ```

  For details, see JVM Memory Settings and System Performance.
- If this doesn't work, try rebooting.

Application or cache server does not join the cluster

Response: Check these possible causes.

- Network problem—the most common cause. First, try to ping the other hosts.
- Firewall problems. If members of your distributed VMware GemFire system are located outside the LAN, check whether the firewall is blocking communication. VMware GemFire is a network-centric distributed system, so if you have a firewall running on your machine, it

could cause connection problems. For example, your connections may fail if your firewall places restrictions on inbound or outbound permissions for Java-based sockets. You may need to modify your firewall configuration to permit traffic to Java applications running on your machine. The specific configuration depends on the firewall you are using.

- Wrong multicast port when using multicast for membership. Check the `gemfire.properties` file of this application or cache server to see that the mcast-port is configured correctly. If you are running multiple clusters at your site, each cluster must use a unique multicast port.
- Can not connect to locator (when using TCP for discovery).
    - Check that the locators attribute in this process's `gemfire.properties` has the correct IP address for the locator.
    - Check that the locator process is running. If not, see instructions for related problem, Data distribution has stopped, although member processes are running.
    - Bind address set incorrectly on a multi-homed host. When you specify the bind address, use the IP address rather than the host name. Sometimes multiple network adapters are configured with the same hostname. See Topology and Communication General Concepts for more information about using bind addresses.
- Wrong version of VMware GemFire . A version mismatch can cause the process to hang or crash. Check the software version with the gemfire version command.

Member process seems to hang

Response:

- **During initialization**—For persistent regions, the member may be waiting for another member with more recent data to start and load from its disk stores. See Disk Storage. Wait for the initialization to finish or time out. The process could be busy—some caches have millions of entries, and they can take a long time to load. Look for this especially with cache servers, because their regions are typically replicas and therefore store all the entries in the region. Applications, on the other hand, typically store just a subset of the entries. For partitioned regions, if the initialization eventually times out and produces an exception, the system architect needs to repartition the data.
- **For a running process**—Investigate whether another member is initializing. Under some optional cluster configurations, a process can be required to wait for a response from other processes before it proceeds.

Member process does not read settings from the gemfire.properties file

Either the process can't find the configuration file or, if it is an application, it may be doing programmatic configuration.

Response:

- Check that the `gemfire.properties` file is in the right directory.
- Make sure the process is not picking up settings from another `gemfire.properties` file earlier in the search path. VMware GemFire looks for a `gemfire.properties` file in the

current working directory, the home directory, and the CLASSPATH, in that order.

- For an application, check the documentation to see whether it does programmatic configuration. If so, the properties that are set programmatically cannot be reset in a `gemfire.properties` file. See your application's customer support group for configuration changes.

Cache creation fails - must match schema definition root

System member startup fails with an error like one of these:

```
Exception in thread "main" org.apache.geode.cache.CacheXmlException:
While reading Cache XML file:/C:/gemfire/client_cache.xml.
Error while parsing XML, caused by org.xml.sax.SAXParseException:
Document root element "client-cache", must match DOCTYPE root "cache".
```

```
Exception in thread "main" org.apache.geode.cache.CacheXmlException:
While reading Cache XML file:/C:/gemfire/cache.xml.
Error while parsing XML, caused by org.xml.sax.SAXParseException:
Document root element "cache", must match DOCTYPE root "client-cache".
```

VMware GemFire declarative cache creation uses one of two root element pairs: `cache` or `client-cache`. The name must be the same in both places.

Response:

- Modify your `cache.xml` file so it has the proper XML namespace and schema definition.

For peers and servers:

```
<?xml version="1.0" encoding="UTF-8"?>
<cache
    xmlns="http://geode.apache.org/schema/cache"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://geode.apache.org/schema/cache http://geode.apache.org/s
chema/cache/cache-1.0.xsd"
    version="1.0”>
...
</cache>
```

For clients:

```
<?xml version="1.0" encoding="UTF-8"?>
<client-cache
    xmlns="http://geode.apache.org/schema/cache"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://geode.apache.org/schema/cache http://geode.apache.org/s
chema/cache/cache-1.0.xsd"
    version="1.0">
...
</client-cache>
```

Cache is not configured properly

An empty cache can be a normal condition. Some applications start with an empty cache and populate it programmatically, but others are designed to bulk load data during initialization.

Response:

If your application should start with a full cache but it comes up empty, check these possible causes:

- **No regions**—If the cache has no regions, the process isn't reading the cache configuration file. Check that the name and location of the cache configuration file match those configured in the cache-xml-file attribute in `gemfire.properties`. If they match, the process may not be reading `gemfire.properties`. See Member process does not read settings from the gemfire.properties file.
- **Regions without data**—If the cache starts with regions, but no data, this process may not have joined the correct cluster. Check the log file for messages that indicate other members. If you don't see any, the process may be running alone in its own cluster. In a process that is clearly part of the correct cluster, regions without data may indicate an implementation design error.

Unexpected results for keySetOnServer and containsKeyOnServer

Client calls to keySetOnServer and containsKeyOnServer can return incomplete or inconsistent results if your server regions are not configured as partitioned or replicated regions.

A non-partitioned, non-replicate server region may not hold all data for the distributed region, so these methods would operate on a partial view of the data set.

In addition, the client methods use the least loaded server for each method call, so may use different servers for two calls. If the servers do not have a consistent view in their local data set, responses to client requests will vary.

The consistent view is only guaranteed by configuring the server regions with partitioned or replicate data-policy settings. Non-server members of the server system can use any allowable configuration as they are not available to take client requests.

The following server region configurations give inconsistent results. These configurations allow different data on different servers. There is no additional messaging on the servers, so no union of keys across servers or checking other servers for the key in question.

- Normal
- Mix (replicated, normal, empty) for a single distributed region. Inconsistent results depending on which server the client sends the request to

These configurations provide consistent results:

- Partitioned server region
- Replicated server region
- Empty server region: keySetOnServer returns the empty set and containsKeyOnServer returns false

Response: Use a partitioned or replicate data-policy for your server regions. This is the only way to provide a consistent view to clients of your server data set. See Region Data Storage and Distribution Options.

Data operation returns PartitionOfflineException

In partitioned regions that are persisted to disk, if you have any members offline, the partitioned region will still be available but may have some buckets represented only in offline disk stores. In this case, methods that access the bucket entries return a PartitionOfflineException, similar to this:

```
org.apache.geode.cache.persistence.PartitionOfflineException:
Region /__PR/_B__root_partitioned__region_7 has persistent data that is no
longer online stored at these locations:
[/192.0.2.1:/export/straw3/users/jpearson/bugfix_Apr10/testCL/hostB/backupDirectory
created at timestamp 1270834766733 version 0]
```

Response: Bring the missing member online, if possible. This restores the buckets to memory and you can work with them again. If the missing member cannot be brought back online, or the disk stores for the member are corrupt, you may need to revoke the member, which will allow the system to create the buckets in new members and resume operations with the entries. See Handling Missing Disk Stores.

Entries are not being evicted or expired as expected

Check these possible causes.

- Transactions—Entries that are due to be expired may remain in the cache if they are involved in a transaction. Further, transactions never time out, so if a transaction hangs, the entries involved in the transaction will remain stuck in the cache. If you have a process with a hung transaction, you may need to end the process to remove the transaction. In your application programming, do not leave transactions open ended. Program all transactions to end with a commit or a rollback.
- Partitioned regions—For performance reasons, eviction and expiration behave differently in partitioned regions and can cause entries to be removed before you expect. See Eviction and Expiration.

Cannot find the log file

Operating without a log file can be a normal condition, so the process does not log a warning.

Response:

- Check whether the log-file attribute is configured in `gemfire.properties`. If not, logging defaults to standard output, and on Windows it may not be visible at all.
- If log-file is configured correctly, the process may not be reading `gemfire.properties`. See Member process does not read settings from the gemfire.properties file.

OutOfMemoryError

An application gets an OutOfMemoryError if it needs more object memory than the process is able to give. The messages include java.lang.OutOfMemoryError.

Response:

The process may be hitting its virtual address space limits. The virtual address space has to be large enough to accommodate the heap, code, data, and dynamic link libraries (DLLs).

- If your application is out of memory frequently, you may want to profile it to determine the cause.
- If you suspect your heap size is set too low, you can increase direct memory by resetting the maximum heap size, using -Xmx. For details, see JVM Memory Settings and System Performance.
- You may need to lower the thread stack size. The default thread stack size is quite large: 512kb on Sparc and 256kb on Intel for 1.3 and 1.4 32-bit JVMs, 1mb with the 64-bit Sparc 1.4 JVM; and 128k for 1.2 JVMs. If you have thousands of threads then you might be wasting a significant amount of stack space. If this is your problem, the error may be this:

  ```
  OutOfMemoryError: unable to create new native thread
  ```

  The minimum setting in 1.3 and 1.4 is 64kb, and in 1.2 is 32kb. You can change the stack size using the -Xss flag, like this: -Xss64k
- You can also control memory use by setting entry limits for the regions.

PartitionedRegionDistributionException

The org.apache.geode.cache.PartitionedRegionDistributionException appears when VMware GemFire fails after many attempts to complete a distributed operation. This exception indicates that no data store member can be found to perform a destroy, invalidate, or get operation.

Response:

- Check the network for traffic congestion or a broken connection to a member.
- Look at the overall installation for problems, such as operations at the application level set to a higher priority than the VMware GemFire processes.
- If you keep seeing PartitionedRegionDistributionException, you should evaluate whether you need to start more members.

PartitionedRegionStorageException

The org.apache.geode.cache.PartitionedRegionStorageException appears when VMware GemFire can't create a new entry. This exception arises from a lack of storage space for put and create operations or for get operations with a loader. PartitionedRegionStorageException often indicates data loss or impending data loss.

The text string indicates the cause of the exception, as in these examples:

```
Unable to allocate sufficient stores for a bucket in the partitioned region....
```

```
Ran out of retries attempting to allocate a bucket in the partitioned region....
```

Response:

- Check the network for traffic congestion or a broken connection to a member.
- Look at the overall installation for problems, such as operations at the application level set to a higher priority than the VMware GemFire processes.
- If you keep seeing PartitionedRegionStorageException, you should evaluate whether you need to start more members.

Application crashes without producing an exception

If an application crashes without any exception, this may be caused by an object memory problem. The process is probably hitting its virtual address space limits. For details, see OutOfMemoryError.

Response: Control memory use by setting entry limits for the regions.

Timeout alert

If a distributed message does not get a response within a specified time, it sends an alert to signal that something might be wrong with the system member that hasn't responded. The alert is logged in the sender's log as a warning.

A timeout alert can be considered normal.

Response:

- If you're seeing a lot of timeouts and you haven't seen them before, check whether your network is flooded.
- If you see these alerts constantly during normal operation, consider raising the ack-wait-threshold above the default 15 seconds.

Member produces SocketTimeoutException

A client and server produces a SocketTimeoutException when it stops waiting for a response from the other side of the connection and closes the socket. This exception typically happens on the handshake or when establishing a callback connection.

Response:

Increase the default socket timeout setting for the member. This timeout is set separately for the client Pool. For a client/server configuration, adjust the "read-timeout" value as described in <pool> or use the `org.apache.geode.cache.client.PoolFactory.setReadTimeout` method.

Member logs ForcedDisconnectException, Cache and DistributedSystem forcibly closed

A cluster member's Cache and DistributedSystem are forcibly closed by the system membership coordinator if it becomes sick or too slow to respond to heartbeat requests. When this happens, listeners receive RegionDestroyed notification with an opcode of FORCED_DISCONNECT. The VMware GemFire log file for the member shows a ForcedDisconnectException with the message

```
This member has been forced out of the cluster because it did not respond
within member-timeout milliseconds
```

Response:

To minimize the chances of this happening, you can increase the DistributedSystem property member-timeout. Take care, however, as this setting also controls the length of time required to notice a network failure. It should not be set too high.

Members cannot see each other

Suspect a network problem or a problem in the configuration of transport for memory and discovery.

Response:

- Check your network monitoring tools to see whether the network is down or flooded.
- If you are using multi-homed hosts, make sure a bind address is set and consistent for all system members. For details about using bind addresses, see Topology and Communication General Concepts.
- Check that all the applications and cache servers are using the same locator address.

One part of the cluster cannot see another part

This situation can leave your caches in an inconsistent state. In networking circles, this kind of network outage is called the “split brain problem.”

Response:

- Restart all the processes to ensure data consistency.
- Going forward, set up network monitoring tools to detect these kinds of outages quickly.
- Enable network partition detection.

Also see Understanding and Recovering from Network Outages.

Data distribution has stopped, although member processes are running

Suspect a problem with the network, the locator, or the multicast configuration, depending on the transport your cluster is using.

Response:

- Check the health of your system members. Search the logs for this string:

  ```
  Uncaught exception
  ```

An uncaught exception means a severe error, often an OutOfMemoryError. See OutOfMemoryError.

- Check your network monitoring tools to see whether the network is down or flooded.
- If you are using multicast, check whether the existing configuration is no long appropriate for the current network traffic.
- Check whether the locators have stopped. For a list of the locators in use, check the locators property in one of the application `gemfire.properties` files.
    - Restart the locator processes on the same hosts, if possible. The cluster begins normal operation, and data distribution restarts automatically.
    - If a locator must be moved to another host or a different IP address, complete these steps:
        1. Shut down all the members of the cluster in the usual order.
        2. Restart the locator process in its new location.
        3. Edit all the gemfire.properties files to change this locator's IP address in the locators attribute.
        4. Restart the applications and cache servers in the usual order.
- Create a watchdog daemon or service on each locator host to restart the locator process when it stops

Distributed-ack operations take a very long time to complete

This problem can occur in systems with a great number of distributed-no-ack operations. That is, the presence of many no-ack operations can cause ack operation to take a long time to complete.

Response:

For information on alleviating this problem, see Slow distributed-ack Messages.

Slow system performance

Slow system performance is sometimes caused by a buffer size that is too small for the objects being distributed.

Response:

If you are experiencing slow performance and are sending large objects (multiple megabytes), try increasing the socket buffer size settings in your system. For more information, see Socket Communication.

Can't get Windows performance data

Attempting to run performance measurements for VMware GemFire on Windows can produce this error message:

```
Can't get Windows performance data. RegQueryValueEx returned 5
```

This error can occur because incorrect information is returned when a Win32 application calls the ANSI version of RegQueryValueEx Win32 API with HKEY_PERFORMANCE_DATA. This error is described in Microsoft KB article ID 226371 at http://support.microsoft.com/kb/226371/en-us.

Response:

To successfully acquire Windows performance data, you need to verify that you have the proper registry key access permissions in the system registry. In particular, make sure that Perflib in the following registry path is readable (KEY_READ access) by the VMware GemFire process:

```
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Perflib
```

An example of reasonable security on the performance data would be to grant administrators KEY_ALL_ACCESS access and interactive users KEY_READ access. This particular configuration would prevent non-administrator remote users from querying performance data.

See http://support.microsoft.com/kb/310426 and http://support.microsoft.com/kb/146906 for instructions about how to ensure that VMware GemFire processes have access to the registry keys associated with performance.

Java applications on 64-bit platforms hang or use 100% CPU

If your Java applications suddenly start to use 100% CPU, you may be experiencing the leap second bug. This bug is found in the Linux kernel and can severely affect Java programs. In particular, you may notice that method invocations using `Thread.sleep(n)` where `n` is a small number will actually sleep for much longer period of time than defined by the method. To verify that you are experiencing this bug, check the host's `dmesg` output for the following message:

```
[10703552.860274] Clock: inserting leap second 23:59:60 UTC
```

To fix this problem, issue the following commands on your affected Linux machines:

```
prompt> /etc/init.d/ntp stop
prompt> date -s "$(date)"
```

See the following web site for more information:

http://blog.wpkg.org/2012/07/01/java-leap-second-bug-30-june-1-july-2012-fix/

System Failure and Recovery

This section describes alerts for and appropriate responses to various kinds of system failures. It also helps you plan a strategy for data recovery.

If a system member withdraws from the cluster involuntarily because the member, host, or network fails, the other members automatically adapt to the loss and continue to operate. The cluster does not experience any disturbance such as timeouts.

Planning for Data Recovery

In planning a strategy for data recovery, consider these factors:

- Whether the region is configured for data redundancy—partitioned regions only.
- The region's role-loss policy configuration, which controls how the region behaves after a crash or system failure—distributed regions only.
- Whether the region is configured for persistence to disk.
- Whether the region is configured for LRU-based eviction.
- The extent of the failure, whether multiple members or a network outage is involved.
- Your application's specific needs, such as the difficulty of replacing the data and the risk of running with inconsistent data for your application.
- When an alert is generated due to network partition or slow response, indicating that certain processes may, or will, fail.

The rest of this section provides recovery instructions for various kinds system failures.

Network Partitioning, Slow Response, and Member Removal Alerts

When a network partition detection or slow responses occur, these alerts are generated:

- Network Partitioning is Detected
- Member is Taking Too Long to Respond
- No Locators Can Be Found
- Warning Notifications Before Removal
- Member is Forced Out

For information on configuring system members to help avoid a network partition configuration condition in the presence of a network failure or when members lose the ability to communicate to each other, refer to Understanding and Recovering from Network Outages.

Network Partitioning Detected

Alert:

```
Membership coordinator id has declared that a network partition has occurred.
```

Description:

This alert is issued when network partitioning occurs, followed by this alert on the individual member:

Alert:

```
Exiting due to possible network partition event due to loss of {0} cache processes:
{1}
```

Response:

Check the network connectivity and health of the listed cache processes.

Member Taking Too Long to Respond

Alert:

```
15 sec have elapsed while waiting for replies: <ReplyProcessor21 6 waiting for 1 repli
es
from [ent(27130):60333/36743]> on ent(27134):60330/45855 whose current membership
list is: [[ent(27134):60330/45855, ent(27130):60333/36743]]
```

Description:

Member ent(27130):60333/36743 is in danger of being forced out of the cluster because of a suspect-verification failure. This alert is issued at the warning level, after the ack-wait-threshold is reached.

Response:

The operator should examine the process to see if it is healthy. The process ID of the slow responder is 27130 on the machine named ent. The ports of the slow responder are 60333/36743. Look for the string, Starting distribution manager ent:60333/36743, and examine the process owning the log file containing this string.

Alert:

```
30 sec have elapsed while waiting for replies: <ReplyProcessor21 6 waiting for 1 repli
es
from [ent(27130):60333/36743]> on ent(27134):60330/45855 whose current membership
list is: [[ent(27134):60330/45855, ent(27130):60333/36743]]
```

Description:

Member ent(27134) is in danger of being forced out of the cluster because of a suspect-verification failure. This alert is issued at the severe level, after the ack-wait-threshold is reached and after ack-severe-alert-threshold seconds have elapsed.

Response:

The operator should examine the process to see if it is healthy. The process ID of the slow responder is 27134 on the machine named ent. The ports of the slow responder are 60333/36743. Look for the string, Starting distribution manager ent:60333/36743, and examine the process owning the log file containing this string.

Alert:

```
15 sec have elapsed while waiting for replies: <DLockRequestProcessor 33636 waiting
for 1 replies from [ent(4592):33593/35174]> on ent(4592):33593/35174 whose current
membership list is: [[ent(4598):33610/37013, ent(4611):33599/60008,
ent(4592):33593/35174, ent(4600):33612/33183, ent(4593):33601/53393, ent(4605):33605/4
1831]]
```

Description:

This alert is issued by partitioned regions and regions with global scope at the warning level, when the lock grantor has not responded to a lock request within the ack-wait-threshold and the ack-severe-alert-threshold.

Response:

None.

Alert:

```
30 sec have elapsed while waiting for replies: <DLockRequestProcessor 23604 waiting
for 1 replies from [ent(4592):33593/35174]> on ent(4598):33610/37013 whose current
membership list is: [[ent(4598):33610/37013, ent(4611):33599/60008,
ent(4592):33593/35174, ent(4600):33612/33183, ent(4593):33601/53393, ent(4605):33605/4
1831]]
```

Description:

This alert is issued by partitioned regions and regions with global scope at the severe level, when the lock grantor has not responded to a lock request within the ack-wait-threshold and the ack-severe-alert-threshold.

Response:

None.

Alert:

```
30 sec have elapsed waiting for global region entry lock held by ent(4600):33612/33183
```

Description

This alert is issued by regions with global scope at the severe level, when the lock holder has held the desired lock for ack-wait-threshold + ack-severe-alert-threshold seconds and may be unresponsive.

Response:

None.

Alert:

```
30 sec have elapsed waiting for partitioned region lock held by ent(4600):33612/33183
```

Description:

This alert is issued by partitioned regions at the severe level, when the lock holder has held the desired lock for ack-wait-threshold + ack-severe-alert-threshold seconds and may be unresponsive.

Response:

None.

No Locators Can Be Found

Note: It is likely that all processes using the locators will exit with the same message.

Alert:

```
Membership service failure: Channel closed: org.apache.geode.ForcedDisconnectExceptio
n:
There are no processes eligible to be group membership coordinator
(last coordinator left view)
```

Description:

Network partition detection is enabled, and there are locator problems.

Response:

The operator should examine the locator processes and logs, and restart the locators.

Alert:

```
Membership service failure: Channel closed: org.apache.geode.ForcedDisconnectExceptio
n:
There are no processes eligible to be group membership coordinator
(all eligible coordinators are suspect)
```

Description:

Network partition detection is enabled, and there are locator problems.

Response:

The operator should examine the locator processes and logs, and restart the locators.

Alert:

```
Membership service failure: Channel closed: org.apache.geode.ForcedDisconnectExceptio
n:
Unable to contact any locators and network partition detection is enabled
```

Description:

Network partition detection is enabled, and there are locator problems.

Response:

The operator should examine the locator processes and logs, and restart the locators.

Alert:

```
Membership service failure: Channel closed: org.apache.geode.ForcedDisconnectExceptio
n:
Disconnected as a slow-receiver
```

Description:

The member was not able to process messages fast enough and was forcibly disconnected by another process.

Response:

The operator should examine and restart the disconnected process.

Warning Notifications Before Removal

Alert:

```
Membership: requesting removal of ent(10344):21344/24922 Disconnected as a slow-receiv
er
```

Description:

This alert is generated only if the slow-receiver functionality is being used.

Response:

The operator should examine the locator processes and logs.

Alert:

```
Network partition detection is enabled and both membership coordinator and lead member
are on the same machine
```

Description:

This alert is issued if both the membership coordinator and the lead member are on the same machine.

Response:

The operator can turn this off by setting the system property gemfire.disable-same-machine-warnings to true. However, it is best to run locator processes, which act as membership coordinators when network partition detection is enabled, on separate machines from cache processes.

Member Is Forced Out

Alert:

```
Membership service failure: Channel closed: org.apache.geode.ForcedDisconnectExceptio
n:
This member has been forced out of the Distributed System. Please consult GemFire logs
to
find the reason.
```

Description:

The process discovered that it was not in the cluster and cannot determine why it was removed. The membership coordinator removed the member after it failed to respond to an internal are-you-alive message.

Response:

The operator should examine the locator processes and logs.

How Data is Recovered From Persistent Regions

A persistent region is one whose contents (keys and values) can be restored from disk. Upon restart, data recovery of a persistent region always recovers keys. Under the default behavior, the region is regarded as ready for use when the keys have been recovered.

The default behavior for restoring values depends on whether the region was configured with an LRU-based eviction algorithm:

- If the region **was not** configured for LRU-based eviction, the values are loaded asynchronously on a separate thread. The assumption here is that all of the stored values will fit into the space allocated for the region.
- If the region **was** configured for LRU-based eviction, the values are not loaded. Each value will be retrieved only when requested. The assumption here is that the values resident in

the region plus any evicted values might exceed the space allocated for the region, possibly resulting in an `OutOfMemoryException` during recovery. **Note:** Recovered values do not contain usage history—LRU history is reset at recovery time.

This default behavior works well under most circumstances. For special cases, three Java system properties allow the developer to modify the recovery behavior for persistent regions:

- `gemfire.disk.recoverValues`

  Default = `true`, recover values for non-LRU regions. Enables the possibility of recovering values for LRU regions (with the setting of an additional property). If `false`, recover only keys, do not recover values. The `false` setting disallows value recovery for LRU regions as well as non-LRU regions.

  How used: When `true`, recovery of the values "warms up" the cache so data retrievals will find their values in the cache, without causing time consuming disk accesses. When `false`, shortens recovery time so the system becomes available for use sooner, but the first retrieval on each key will require a disk read.

- `gemfire.disk.recoverLruValues`

  Default = `false`, do not recover values for a region configured with LRU-based eviction. If `true`, recover all of the LRU region's values. **Note:** `gemfire.disk.recoverValues` must also be `true` for this property to take effect.

  How used: When `false`, shortens recovery time for an LRU-configured region by not loading values. When `true`, restores data values to the cache. As stated above, LRU history is not recoverable, and recovering values for a region configured with LRU-based eviction incurs some risk of exceeding allocated memory.

- `gemfire.disk.recoverValuesSync`

  Default = `false`, recover values by an asynchronous background process. If `true`, values are recovered synchronously, and recovery is not complete until all values have been retrieved. **Note:** `gemfire.disk.recoverValues` must also be `true` for this property to take effect.

  How used: When `false`, allows the system to become available sooner, but some time must elapse before all values have been read from disk into cache memory. Some key retrievals will require disk access, and some will not. When `true`, prolongs restart time, but ensures that when available for use, the cache is fully populated and data retrieval times will be optimal.

Handling Forced Cache Disconnection Using Autoreconnect

A VMware GemFire member may be forcibly disconnected from a VMware GemFire cluster if the member is unresponsive for a period of time, or if a network partition separates one or more members into a group that is too small to act as the cluster.

How the Autoreconnection Process Works

After being disconnected from a cluster, a VMware GemFire member shuts down and, by default, automatically restarts into a "reconnecting" state, while periodically attempting to rejoin the cluster by contacting a list of known locators. If the member succeeds in reconnecting to a known locator, the member rebuilds its view of the cluster from existing members and receives a new distributed member ID.

If the member cannot connect to a known locator, the member will then check to see if it itself is a locator (or hosting an embedded locator process). If the member is a locator, then the member does a quorum-based reconnect; it will attempt to contact a quorum of the members that were in the membership view just before it became disconnected. If a quorum of members can be contacted, then startup of the cluster is allowed to begin. Since the reconnecting member does not know which members survived the network partition event, all members that are in a reconnecting state will keep their UDP unicast ports open and respond to ping requests.

Membership quorum is determined using the same member weighting system used in network partition detection. See Membership Coordinators, Lead Members and Member Weighting.

Note that when a locator is in the reconnecting state, it provides no discovery services for the cluster.

The default settings for reconfiguration of the cache once reconnected assume that the cluster configuration service has a valid (XML) configuration. This will not be the case if the cluster was configured using API calls. To handle this case, either disable autoreconnect by setting the property to

```
disable-auto-reconnect = true
```

or, disable the cluster configuration service by setting the property to

```
enable-cluster-configuration = false
```

After the cache has reconnected, applications must fetch a reference to the new Cache, Regions, DistributedSystem and other artifacts. Old references will continue to throw cancellation exceptions like `CacheClosedException(cause=ForcedDisconnectException)`.

See the VMware GemFire `DistributedSystem` and `Cache` Java API documentation for more information.

Managing the Autoreconnection Process

By default a VMware GemFire member will try to reconnect until it is told to stop by using the `DistributedSystem.stopReconnecting()` or `Cache.stopReconnecting()` method. You can disable automatic reconnection entirely by setting `disable-auto-reconnect` VMware GemFire property to “true.”

You can use `DistributedSystem` and `Cache` callback methods to perform actions during the reconnect process, or to cancel the reconnect process if necessary.

The `DistributedSystem` and `Cache` API provide several methods you can use to take actions while a member is reconnecting to the cluster:

- `DistributedSystem.isReconnecting()` returns true if the member is in the process of reconnecting and recreating the cache after having been removed from the system by other members.

- `DistributedSystem.waitUntilReconnected(long, TimeUnit)` waits for a period of time, and then returns a boolean value to indicate whether the member has reconnected to the DistributedSystem. Use a value of -1 seconds to wait indefinitely until the reconnect completes or the member shuts down. Use a value of 0 seconds as a quick probe to determine if the member has reconnected.
- `DistributedSystem.getReconnectedSystem()` returns the reconnected DistributedSystem.
- `DistributedSystem.stopReconnecting()` stops the reconnection process and ensures that the DistributedSystem stays in a disconnected state.
- `Cache.isReconnecting()` returns true if the cache is attempting to reconnect to a cluster.
- `Cache.waitUntilReconnected(long, TimeUnit)` waits for a period of time, and then returns a boolean value to indicate whether the DistributedSystem has reconnected. Use a value of -1 seconds to wait indefinitely until the reconnect completes or the cache shuts down. Use a value of 0 seconds as a quick probe to determine if the member has reconnected.
- `Cache.getReconnectedCache()` returns the reconnected Cache.
- `Cache.stopReconnecting()` stops the reconnection process and ensures that the DistributedSystem stays in a disconnected state.

Operator Intervention

You may need to intervene in the autoreconnection process if processes or hardware have crashed or are otherwise shut down before the network connection is healed. In this case the members in a “reconnecting” state will not be able to find the lost processes through UDP probes and will not rejoin the system until they are able to contact a locator.

Recovering from Application and Cache Server Crashes

When the application or cache server crashes, its local cache is lost, and any resources it owned (for example, distributed locks) are released. The member must recreate its local cache upon recovery.

- **Recovering from Crashes with a Peer-to-Peer Configuration**

  When a member crashes, the remaining members continue operation as though the missing application or cache server had never existed. The recovery process differs according to region type and scope, as well as data redundancy configuration.

- **Recovering from Crashes with a Client/Server Configuration**

  In a client/server configuration, you first make the server available as a member of a cluster again, and then restart clients as quickly as possible. The client recovers its data from its servers through normal operation.

Recovering from Crashes with a Peer-to-Peer Configuration

When a member crashes, the remaining members continue operation as though the missing application or cache server had never existed. The recovery process differs according to region type and scope, as well as data redundancy configuration.

The other system members are told that it has left unexpectedly. If any remaining system member is waiting for a response (ACK), the ACK still succeeds and returns, because every member that is still alive has responded. If the lost member had ownership of a GLOBAL entry, then the next attempt to obtain that ownership acts as if no owner exists.

Recovery depends on how the member has its cache configured. This section covers the following:

- Recovery for Partitioned Regions
- Recovery for Distributed Regions
- Recovery for Regions of Local Scope
- Recovering Data From Disk

To tell whether the regions are partitioned, distributed, or local, check the `cache.xml` file. If the file contains a local scope setting, the region has no connection to any other member:

```
<region-attributes scope="local">
```

If the file contains any other scope setting, it is configuring a distributed region. For example:

```
<region-attributes scope="distributed-no-ack">
```

If the file includes either of the following lines, it is configuring a partitioned region.

```
<partition-attributes...
<region-attributes data-policy="partition"/>
<region-attributes data-policy="persistent-partition"/>
```

The reassigned clients continue operating smoothly, as in the failover case. A successful rebalancing operation does not create any data loss.

If rebalancing fails, the client fails over to an active server with the normal failover behavior.

Recovery for Partitioned Regions

When an application or cache server crashes, any data in local memory is lost, including any entries in a local partitioned region data store.

Recovery for Partitioned Regions With Data Redundancy

If the partitioned region is configured for redundancy and a member crashes, the system continues to operate with the remaining copies of the data. You may need to perform recovery actions depending on how many members you have lost and how you have configured redundancy in your system.

By default, VMware GemFire does not make new copies of the data until a new member is brought online to replace the member that crashed. You can control this behavior using the recovery delay attributes. For more information, see Configure High Availability for a Partitioned Region.

To recover, start a replacement member. The new member regenerates the lost copies and returns them to the configured redundancy level.

Note: Make sure the replacement member has at least as much local memory as the old one— the `local-max-memory` configuration setting must be the same or larger. Otherwise, you can get into a situation where some entries have all their redundant copies but others don't. In addition, until you have restarted a replacement member, any code that attempts to create or update data mapped to partition region bucket copies (primary and secondary) that have been lost can result in an exception. (New transactions unrelated to the lost data can fail as well simply because they happen to map to– or "resolve" to– a common bucketId).

Even with high availability, you can lose data if too many applications and cache servers fail at the same time. Any lost data is replaced with new data created by the application as it returns to active work.

The number of members that can fail at the same time without losing data is equal to the number of redundant copies configured for the region. So if redundant-copies=1, then at any given time only one member can be down without data loss. If a second goes down at the same time, any data stored by those two members will be lost.

You can also lose access to all copies of your data through network failure. See Understanding and Recovering from Network Outages.

Recovery Without Data Redundancy

If a member crashes and there are no redundant copies, any logic that tries to interact with the bucket data is *blocked* until the primary buckets are restored from disk. (If you do not have persistence enabled, VMware GemFire will reallocate the buckets on any available remaining nodes, however you will need to recover any lost data using external mechanisms.)

To recover, restart the member. The application returns to active work and automatically begins to create new data.

If the members with the relevant disk stores cannot be restarted, then you will have to revoke the missing disk stores manually using gfsh. See revoke missing-disk-store.

Maintaining and Recovering Partitioned Region Redundancy

The following alert [ALERT-1] (warning) is generated when redundancy for a partitioned region drops:

Alert:

```
[warning 2008/08/26 17:57:01.679 PDT dataStoregemfire5_jade1d_6424
<PartitionedRegion Message Processor2> tid=0x5c] Redundancy has dropped below 3
configured copies to 2 actual copies for /partitionedRegion
```

```
[warning 2008/08/26 18:13:09.059 PDT dataStoregemfire5_jade1d_6424
<DM-MemberEventInvoker> tid=0x1d5] Redundancy has dropped below 3
configured copies to 1 actual copy for /partitionedRegion
```

The following alert [ALERT-2] (warning) is generated when, after creation of a partitioned region bucket, the program is unable to find enough members to host the configured redundant copies:

Alert:

```
[warning 2008/08/27 17:39:28.876 PDT gemfire_2_4 <RMI TCP Connection(67)-192.0.2.0>
tid=0x1786] Unable to find sufficient members to host a bucket in the partitioned regi
on.
```

```
Region name = /partitionedregion Current number of available data stores: 1 number
successfully allocated = 1 number needed = 2 Data stores available:
[pippin(21944):41927/42712] Data stores successfully allocated:
[pippin(21944):41927/42712] Consider starting another member
```

The following alert [EXCEPTION-1] (warning) and exception is generated when, after the creation of a partitioned region bucket, the program is unable to find any members to host the primary copy:

Alert:

```
[warning 2008/08/27 17:39:23.628 PDT gemfire_2_4 <RMI TCP Connection(66)-192.0.2.0>
tid=0x1888] Unable to find any members to host a bucket in the partitioned region.
Region name = /partitionedregion Current number of available data stores: 0 number
successfully allocated = 0 number needed = 2 Data stores available:
[] Data stores successfully allocated: [] Consider starting another member
```

Exception:

```
org.apache.geode.cache.PartitionedRegionStorageException: Unable to find any members t
o
                    host a bucket in the partitioned region.
```

- Region name = /partitionedregion
- Current number of available data stores: 0
- Number successfully allocated = 0; Number needed = 2
- Data stores available: []
- Data stores successfully allocated: []

Response:

- Add additional members configured as data stores for the partitioned region.
- Consider starting another member.

Recovery for Distributed Regions

Restart the process. The system member recreates its cache automatically. If replication is used, data is automatically loaded from the replicated regions, creating an up-to-date cache in sync with the rest of the system. If you have persisted data but no replicated regions, data is automatically loaded from the disk store files. Otherwise, the lost data is replaced with new data created by the application as it returns to active work.

Recovery for Regions of Local Scope

Regions of local scope have no memory backup, but may have data persisted to disk. If the region is configured for persistence, the data remains in the region’s disk directories after a crash. The data on disk will be used to initialize the region when you restart.

Recovering Data from Disk

When you persist a region, the entry data on disk outlives the region in memory. If the member exits or crashes, the data remains in the region's disk directories. See Disk Storage. If the same region is created again, this saved disk data can be used to initialize the region.

Some general considerations for disk data recovery:

- Region persistence causes only entry keys and values to be stored to disk. Statistics and user attributes are not stored.
- If the application was writing to the disk asynchronously, the chances of data loss are greater. The choice is made at the region level, with the disk-synchronous attribute.
- When a region is initialized from disk, last modified time is persisted from before the member exit or crash. For information on how this might affect the region data, see Expiration.

Disk Recovery for Disk Writing—Synchronous Mode and Asynchronous Mode

Synchronous Mode of Disk Writing

Alert 1:

```
DiskAccessException has occurred while writing to the disk for region <Region_Name>.
Attempt will be made to destroy the region locally.
```

Alert 2:

```
Encountered Exception in destroying the region locally
```

Description:

These are error log-level alerts. Alert 2 is generated only if there was an error in destroying the region. If Alert 2 is not generated, then the region was destroyed successfully. The message indicating the successful destruction of a region is logged at the information level.

Alert 3:

```
Problem in stopping Cache Servers. Failover of clients is suspect
```

Description:

This is an error log-level alert that is generated only if servers were supposed to stop but encountered an exception that prevented them from stopping.

Response:

The region may no longer exist on the member. The cache servers may also have been stopped. Recreate the region and restart the cache servers.

Asynchronous Mode of Disk Writing

Alert 1:

```
Problem in Asynch writer thread for region <Region_name>. It will terminate.
```

Alert 2:

```
Encountered Exception in destroying the region locally
```

Description:

These are error log-level alerts. Alert 2 is generated only if there was an error in destroying the region. If Alert 2 is not generated, then the region was destroyed successfully. The message indicating the successful destruction of a region is logged at the information level.

Alert 3:

```
Problem in stopping Cache Servers. Failover of clients is suspect
```

Description:

This is an error log-level alert that is generated only if servers were supposed to stop but encountered an exception that prevented them from stopping.

Response:

The region may no longer exist on the member. The cache servers may also have been stopped. Recreate the region and restart the cache servers.

Recovering from Crashes with a Client/Server Configuration

In a client/server configuration, you first make the server available as a member of a cluster again, and then restart clients as quickly as possible. The client recovers its data from its servers through normal operation.

How well a client/server configuration recovers from application or cache server crashes depends on server availability and on client configuration. Typically, the servers are made highly available by running enough servers spread out on enough machines to ensure a minimum of coverage in case of network, machine, or server crashes. The clients are usually configured to connect to a primary and some number of secondary, or redundant, servers. The secondaries act as hot backups to the primary. For high availability of messaging in the case of client crashes, the clients may have durable connections to their servers. If this is the case, some or all of their data and data events remain in server memory and are automatically recovered, providing that you restart the clients within a configured timeout. See Configuring Client/Server Event Messaging for information about durable messaging.

Recovering from Server Failure

Recovery from server failure has two parts: the server recovers as a member of a cluster, then its clients recover its services.

When servers fail, their own recovery is carried out as for any member of a cluster as described in Recovering from Crashes with a Peer-to-Peer Configuration.

From the client’s perspective, if the system is configured for high availability, server failure goes undetected unless enough servers fail that the server-to-client ratio drops below a workable level. In any case, your first course of action is to get the servers back up as quickly as possible.

To recover from server failure:

1. Recover the server and its data as described in Recovering from Crashes with a Peer-to-Peer Configuration.
2. Once the server is available again, the locators (or client pools if you are using a static server list) automatically detect its presence and add it to the list of viable servers. It might take awhile for the clients to start using the recovered server. The time depends in part on how the clients are configured and how they are programmed. See Client/Server Configuration.

If you need to start a server at a new host/port location

This section is only for systems where the clients' server pool configurations use static server lists. This is unusual, but might be the case for your system. If the server pools are configured without static server lists, meaning clients use locators to find their servers, starting a server at a new address requires no special action because the new server is automatically detected by the locators. You can determine whether your clients use locator lists or server lists by looking at the client `cache.xml` files. Systems configured with static server lists have <server> elements listed inside the <pool> elements. Those using locator lists have <locator> elements instead. If there are no pools declared in the XML files, the servers or locators will be defined in the application code. Look for the API PoolFactory methods addServer or addLocator.

If the pools are configured with static server lists, the clients only connect to servers at the specific addresses provided in the lists. To move a server or add a server at a new location, you must modify the <server> specifications in the clients' `cache.xml` file. This change will only affect newly-started clients. To start using the new server information, either restart clients or wait for new clients to start, depending on your system characteristics and how quickly you need the changes to take effect.

Recovering from Client Failure

When a client crashes, restart it as quickly as possible in the usual way. The client recovers its data from its servers through normal operation. Some of the data may be recovered immediately, and some may be recovered lazily as the client requests it. Additionally, the server may be configured to replay events for some data and for some client queries. These are the different configurations that affect client recovery:

- **Entries immediately sent to the client**—Entries are immediately sent to the client for entries the client registers interest in, if those entries are present in the server cache.
- **Entries sent lazily to the client**—Entries are sent lazily to the client for entries that the client registers interest in that are not initially available in the server cache.
- **Events sent immediately to the client**—If the server has been saving events for the client, these are immediately replayed when the client reconnects. Cache modification events for entries in which the client has registered durable interest are saved.

If you have a durable client configured to connect to multiple servers, keep in mind that VMware GemFire does not maintain server redundancy while the client is disconnected. If you lose all of its primary and secondary servers, you lose the client's queued messages. Even if the servers fail one at a time, so that running clients have time to fail over and pick new secondary servers, an off-line durable client cannot do that and thus loses its queued messages.

Recovering from Machine Crashes

When a machine crashes because of a shutdown, power loss, hardware failure, or operating system failure, all of its applications and cache servers and their local caches are lost.

System members on other machines are notified that this machine's members have left the cluster unexpectedly.

Recovery Procedure

To recover from a machine crash:

1. Determine which processes run on this machine.
2. Reboot the machine.
3. If a VMware GemFire locator runs here, start it first. **Note:** At least one locator must be running before you start any applications or cache servers.
4. Start the applications and cache servers in the usual order.

If you have to move a locator process to a different machine, the locator isn't useful until you update the locators list in the `gemfire.properties` file and restart all the applications and cache servers in the cluster. If other locators are running, however, you don't have to restart the system immediately. For a list of the locators in use, check the locators property in one of the application `gemfire.properties` files.

Data Recovery for Partitioned Regions

The partitioned region initializes itself correctly regardless of the order in which the data stores rejoin. The applications and cache servers recreate their data automatically as they return to active work.

If the partitioned region is configured for data redundancy, VMware GemFire may be able to handle a machine crash automatically with no data loss, depending on how many redundant copies there are and how many members have to be restarted. See also Recovery for Partitioned Regions.

If the partitioned region does not have redundant copies, the system members recreate the data through normal operation. If the member that crashed was an application, check whether it was designed to write its data to an external data source. If so, decide whether data recovery is possible and preferable to starting with new data generated through the VMware GemFire cluster.

Data Recovery for Distributed Regions

The applications and cache servers recreate their data automatically. Recovery happens through replicas, disk store files, or newly generated data, as explained in Recovery for Distributed Regions.

If the recovery is from disk stores, you may not get all of the latest data. Persistence depends on the operating system to write data to the disk, so when the machine or operating system fails unexpectedly, the last changes can be lost.

For maximum data protection, you can set up duplicate replicate regions on the network, with each one configured to back up its data to disk. Assuming the proper restart sequence, this architecture significantly increases your chances of recovering every update.

Data Recovery in a Client/Server Configuration

If the machine that crashed hosted a server, how the server recovers its data depends on whether the regions are partitioned or distributed. See Data Recovery for Partitioned Regions and Data Recovery for Distributed Regions as appropriate.

The impact of a server crash on its clients depends on whether the installation is configured for highly available servers. For information, see Recovering from Crashes with a Client/Server Configuration.

If the machine that crashed hosted a client, restart the client as quickly as possible and let it recover its data automatically from the server. For details, see Recovering from Client Failure.

Recovering from ConfictingPersistentDataExceptions

A `ConflictingPersistentDataException` while starting up persistent members indicates that you have multiple copies of some persistent data, and VMware GemFire cannot determine which copy to use.

Normally VMware GemFire uses metadata to determine automatically which copy of persistent data to use. Along with the region data, each member persists a list of other members that are hosting the region and whether their data is up to date. A `ConflictingPersistentDataException` happens when two members compare their metadata and find that it is inconsistent. The members either don't know about each other, or they both think the other member has stale data.

The following sections describe scenarios that can cause `ConflictingPersistentDataException`s in VMware GemFire and how to resolve the conflict.

Independently Created Copies

Trying to merge two independently created clusters into a single cluster will cause a `ConflictingPersistentDataException`.

There are a few ways to end up with independently created systems.

- Create two different clusters by having members connect to different locators that are not aware of each other.
- Shut down all persistent members and then start up a different set of brand new persistent members.

VMware GemFire will not automatically merge independently created data for the same region. Instead, you need to export the data from one of the systems and import it into the other system. See the section Cache and Region Snapshots for instructions on how to export data from one system and import it into another.

Starting New Members First

Starting a brand new member that has no persistent data before starting older members with persistent data can cause a `ConflictingPersistentDataException`.

One accidental way this can happen is to shut the system down, add a new member to the startup scripts, and start all members in parallel. By chance, the new member may start first. The issue is that the new member will create an empty, independent copy of the data before the older members start up. VMware GemFire will be treat this situation like the Independently Created Copies case.

In this case the fix is simply to move aside or delete the persistent files for the new member, shut down the new member and then restart the older members. When the older members have fully recovered, then restart the new member.

A Network Failure Occurs and Network Partitioning Detection is Disabled

When `enable-network-partition-detection` is set to the default value of true, VMware GemFire will detect a network partition and shut down unreachable members to prevent a network partition ("split brain") from occurring. No conflicts should occur when the system is healed.

However if `enable-network-partition-detection` is false, VMware GemFire will not detect the network partition. Instead, each side of the network partition will end up recording that the other side of the partition has stale data. When the partition is healed and persistent members are restarted, the members will report a conflict because both sides of the partition think the other members are stale.

In some cases it may be possible to choose between sides of the network partition and just keep the data from one side of the partition. Otherwise you may need to salvage data and import it into a fresh system.

Salvaging Data

If you receive a `ConflictingPersistentDataException`, you will not be able to start all of your members and have them join the same cluster. You have some members with conflicting data.

First, see if there is part of the system that you can recover. For example if you just added some new members to the system, try to start up without including those members.

For the remaining members you can extract data from the persistent files on those members and import the data.

To extract data from the persistent files, use the `gfsh export offline-disk-store` command.

```
gfsh> export offline-disk-store --name=MyDiskStore --disk-dirs=./mydir --dir=./outputd
ir
```

This will produce a set of snapshot files. Those snapshot files can be imported into a running system using:

```
gfsh> import data --region=/myregion --file=./outputdir/snapshot-snapshotTest-test0.gf
d --member=server1
```

Preventing and Recovering from Disk Full Errors

It is important to monitor the disk usage of VMware GemFire members. If a member lacks sufficient disk space for a disk store, the member attempts to shut down the disk store and its associated cache, and logs an error message. A shutdown due to a member running out of disk space can cause loss of data, data file corruption, log file corruption and other error conditions that can negatively impact your applications.

After you make sufficient disk space available to the member, you can restart the member.

You can prevent disk file errors using the following techniques:

- If you are using ext4 file system, we recommend that you pre-allocate disk store files and disk store metadata files. Pre-allocation reserves disk space for these files and leaves the member in a healthy state when the disk store and regions are shut down, allowing you to restart the member once sufficient disk space has been made available. Pre-allocation is enabled by default.
- Configure critical usage thresholds (disk-usage-warning-percentage and disk-usage-critical-percentage) for the disk. By default, these are set to 90% for warning and 99% for errors that will shut down the cache.
- Follow the recommendations in Optimizing a System with Disk Stores for general disk management best practices.

When a disk write fails due to disk full conditions, the member is shutdown and removed from the cluster.

Recovering from Disk Full Errors

If a member of your cluster fails due to a disk full error condition, add or make additional disk capacity available and attempt to restart the member normally. If the member does not restart and there is a redundant copy of its regions in a disk store on another member, you can restore the member using the following steps:

1. Delete or move the disk store files from the failed member.
2. Use the gfsh `show missing-disk-stores` command to identify any missing data. You may need to manually restore this data.
3. Revoke the missing disk stores using the revoke missing-disk-store gfsh command.
4. Restart the member.

See Handling Missing Disk Stores for more information.

Understanding and Recovering from Network Outages

The safest response to a network outage is to restart all the processes and bring up a fresh data set.

However, if you know the architecture of your system well, and you are sure you won’t be resurrecting old data, you can do a selective restart. At the very least, you must restart all the members on one side of the network failure, because a network outage causes separate clusters that can’t rejoin automatically.

What Happens During a Network Outage

When the network connecting members of a cluster goes down, system members treat this like a machine crash. Members on each side of the network failure respond by removing the members on the other side from the membership list. If network partitioning detection is enabled (the default), the partition that contains sufficient quorum (> 51% based on member weight) will continue to operate, while the other partition with insufficient quorum will shut down. See Network Partitioning for a detailed explanation on how this detection system operates.

In addition, members that have been disconnected either via network partition or due to unresponsiveness will automatically try to reconnect to the cluster unless configured otherwise. See Handling Forced Cache Disconnection Using Autoreconnect.

Recovery Procedure

For deployments that have network partition detection and/or auto-reconnect disabled, to recover from a network outage:

1. Decide which applications and cache servers to restart, based on the architecture of the cluster. Assume that any process other than a data source is bad and needs restarting. For example, if an outside data feed is coming in to one member, which then redistributes to all the others, you can leave that process running and restart the other members.
2. Shut down all the processes that need restarting.
3. Restart them in the usual order.

The members recreate the data as they return to active work. For details, see Recovering from Application and Cache Server Crashes.

Effect of Network Failure on Partitioned Regions

Both sides of the cluster continue to run as though the members on the other side were not running. If the members that participate in a partitioned region are on both sides of the network failure, both sides of the partitioned region also continue to run as though the data stores on the other side did not exist. In effect, you now have two partitioned regions.

When the network recovers, the members may be able to see each other again, but they are not able to merge back together into a single cluster and combine their buckets back into a single partitioned region. You can be sure that the data is in an inconsistent state. Whether you are configured for data redundancy or not, you don't really know what data was lost and what wasn't. Even if you have redundant copies and they survived, different copies of an entry may have different values reflecting the interrupted workflow and inaccessible data.

Effect of Network Failure on Distributed Regions

By default, both sides of the cluster continue to run as though the members on the other side were not running. For distributed regions, however, the regions's reliability policy configuration can change this default behavior.

When the network recovers, the members may be able to see each other again, but they are not able to merge back together into a single cluster.

Effect of Network Failure on Persistent Regions

A network failure when using persistent regions can cause conflicts in your persisted data. When you recover your system, you will likely encounter `ConflictingPersistentDataException`s when members start up.

For this reason, `enable-network-partition-detection` must be set to true if you are using persistent regions.

For information on how to recover from `ConflictingPersistentDataException` errors should they occur, see Recovering from ConfictingPersistentDataExceptions.

Effect of Network Failure on Client/Server Installations

If a client loses contact with all of its servers, the effect is the same as if it had crashed. You need to restart the client. See Recovering from Client Failure. If a client loses contact with some servers, but not all of them, the effect on the client is the same as if the unreachable servers had crashed. See Recovering from Server Failure.

Servers, like applications, are members of a cluster, so the effect of network failure on a server is the same as for an application. Exactly what happens depends on the configuration of your site.

Log Messages and Solutions

This section provides explanations of VMware GemFire Log messages with potential resolutions.

Depending on how your system is configured, log files can be found in a number of locations. See Log File Locations and Naming, Searching, and Creating Log Files for more information.

above heap eviction threshold

Log Message:

```
[info 2021/03/23 16:00:13.721 EDT xxx-server01 <Notification Handler> tid=0x5d] Membe
r:
xxx(xxx-server01:29847)<v9>:11096 above heap eviction threshold
```

Log Level: info

Category: Heap GC

Meaning:

This message requires action to remain healthy. The live objects are driving heap consumption above your threshold for collecting heap. This is not a good state, as you will either be prematurely destroying data or overflowing it to disk, which can overwhelm the disk.

Potential Resolutions:

NOTE: VMware GemFire eviction is not truly compatible with G1GC given how G1GC behaves and how eviction assumes that garbage will be collected.

You should consider increasing the total heap. This will increase tenured space, and potentially eliminate these messages. You can also increase your eviction-threshold percentage, but this can risk growing heap to the point where you encounter heap fragmentation issues.

below heap eviction threshold

Log Message:

```
[info 2021/03/23 16:00:43.438 EDT xxx-server01 <Notification Handler> tid=0x5d] Membe
r:
xxx(xxx-server01:29847)<v9>:11096 below heap eviction threshold
```

Log Level: info

Category: Heap GC

Meaning:

You are now below the eviction threshold, after having been above the threshold.

Potential Resolutions:

Follow the guidance provided in the “above heap eviction threshold” message.

above heap critical threshold

Log Message:

```
[error 2020/06/23 03:43:48.796 EDT <Notification Handler1> tid=0xa4] Member:
xxx(xxx-server-2:119506)<v2>:10102 above heap critical threshold. Event generated via
polling. Used bytes: 26508001280. Memory thresholds: MemoryThresholds@[2041517395
maxMemoryBytes:26843545600, criticalThreshold:95.0,
criticalThresholdBytes:25501368320, criticalThresholdClearBytes:24964497408,
evictionThreshold:85.0, evictionThresholdBytes:22817013760,
evictionThresholdClearBytes:22280142848]
```

Log Level: error

Category: Heap GC

Meaning:

This message requires **URGENT** action. You are in danger of VMware GemFire distributed system issues where a member, or members, may be kicked out with potential major business impact. The live objects are driving heap consumption above your critical threshold, so either garbage collection is proving ineffective or your usage has increased unexpectedly, taking you to much higher levels of heap consumption. Take action **immediately** if you ever see this, even if you were not negatively impacted at the time.

Potential Resolutions:

If you do not already have VMware GemFire eviction in place, acting as a level of protection to keep heap consumption lower, consider incorporating some flavor of eviction. G1GC and other newer collectors are not really compatible with HEAP_LRU eviction, so you would need to incorporate entry count or memory-based eviction.

Generally, being above the critical threshold means that you likely need to increase your total heap. If you have the critical-threshold set relatively low given your heap size, you could consider increasing this value. Having a critical-threshold of 90%, for example, with a 30g heap is too low. This is essentially wasting 3g of heap acting purely as overhead protection.

The recommendation is to set the critical-threshold to see the percentage high enough such that you have a maximum of 1g of overhead. This means that setting the critical-threshold to 98 would be completely fine for a 100g heap. If you are seeing tenured heap growth with no entry count growth over time, this is likely indicative of a leak. You will need to take heap dumps and analyze them to determine why the heap is growing. It could be only temporary, if queries are running and driving heap consumption, but this should resolve itself, since VMware GemFire will terminate queries and eliminate that garbage.

If you are using G1GC, it is possible that you are not setting your InitiatingHeapOccupancyPercent low enough. The default of 45 is too high, so consider trying 30% to see if the tenured heap behavior becomes more stable.

Query execution canceled after exceeding max execution time

Log Message:

```
[info 2021/02/05 03:56:08.087 EST xxx<QueryMonitor Thread> tid=0x3d9] Query execution
canceled after exceeding max execution time 600000ms. Query String = SELECT * FROM
/xxx);isCancelled = true; Total Executions = 3045391; Total Execution Time = 0
```

Log Level: info

Category: Operations

Meaning:

The query is taking longer than the configured execution time (600,000ms, in this example). Perhaps it was a rogue query. Perhaps you are short of the system resources needed to accommodate the current level of activity, including this query.

Potential Resolutions:

If this persists over time, then the query is likely taking too long independent of the current system state, so you may need to increase the configured time by setting the VMware GemFire system property, “gemfire.MAX_QUERY_EXECUTION_TIME”, to something higher in order to allow the query to complete. If this property is not set, the query will never timeout unless you are using the resource manager, in which case it will timeout in 5 hours. This property does provide some protection against a really problematic query or set of queries, but requires you to understand what is driving the query times to know how high to set it.

Perhaps the query did not incorporate the use of a configured index, or indexes, for some reason. In order to obtain this deeper understanding, you can incorporate verbose logging for your queries by setting the VMware GemFire system property, “gemfire.Query.VERBOSE”.

Query execution canceled due to memory threshold crossed in system

Log Message:

```
[warning 2018/03/02 09:33:44.516 EST xxx <ServerConnection on port 40401 Thread 24>
tid=0x1a9] Server connection from [identity(xxx(14:loner):x:x,connection=2;
port=33218]: Unexpected Exception
org.apache.geode.cache.query.QueryExecutionLowMemoryException: Query execution
canceled due to memory threshold crossed in system, memory used: 23,540,738,136
bytes.
```

Log Level: warning

Category: Operations

Meaning:

Very self explanatory here. A query was canceled because some member or members have crossed the critical-threshold configured in the system. To protect the member(s) from running out of memory, the query is terminated. The message indicates the number of bytes used at the time, which is certainly more than the number of bytes equating to the critical-threshold percentage, in bytes. You should also see the “above heap critical threshold” message in some logs as well if seeing this message, to understand the problem members.

Potential Resolutions:

The root cause for the heap issues needs to be investigated. Perhaps it is simply the need for more total heap. Perhaps GC activity is not collecting garbage effectively, which happens especially with some G1GC configurations. Perhaps it is a rogue query driving much more new object activity than expected, or running too long such that the tenured heap becomes much more full than normal behavior.

You could increase the critical-threshold to some higher percentage, but that may just delay the inevitable. You could configure your regions to use the eviction-threshold, which will protect the system in many cases of hitting such high levels of heap surpassing the critical-threshold configured.

There are <n> stuck threads in this node

Thread <n> is stuck

Thread <n> that was executed at <time> has been stuck for <nn> seconds

Log Message:

```
[warning 2021/04/06 00:16:51.743 EDT rtp <ThreadsMonitor> tid=0x11] There are <13>
stuck threads in this node
[warning 2021/04/06 00:17:51.737 EDT rtp <ThreadsMonitor> tid=0x11] Thread <51392> is
stuck
[warning 2021/04/06 00:17:51.738 EDT rtp <ThreadsMonitor> tid=0x11] Thread <51392>
that was executed at <06 Apr 2021 00:16:12 EDT> has been stuck for <99.119 seconds>
and number of thread monitor iteration <2>
Thread Name <poolTimer-gemfire-pool-35493>
  Thread state <WAITING>
  Waiting on <java.util.concurrent.locks.ReentrantLock$NonfairSync@cae7911>
  Owned By <Function Execution Thread-2410> and ID <50995>
  Executor Group <ScheduledThreadPoolExecutorWithKeepAlive>
  Monitored metric <ResourceManagerStats.numThreadsStuck>
  Thread Stack:   UNIQUE TO EACH CASE
```

Log Level: warning

Category: Operations

Meaning:

These messages requires **URGENT** action, to determine whether any issues exist. It is very possible that there are no real issues, but it is also possible this is the beginning of a major issue that could snowball to impact the entire cluster. These messages require deeper investigation.

Potential Resolutions:

First, if you only see this issue rarely, or only for a single iteration, it is almost certainly not an issue. The word "stuck" here may be misleading. The messages are saying that it appears that this thread has been doing the same thing for a while, so it may be stuck. Some tasks, such as taking backups, doing exports, or running a rebalance, may appear to be "stuck" when in reality they are simply doing the same thing over and over as it progresses, like moving a bucket. While it may appear that we are still moving buckets, it's probably a different bucket each time.

A key indicator that a thread is truly stuck is the number of iterations, as indicated in the "has been stuck" message above. If you know that the operation is not one that should take so long, and you see an iteration of <10> or higher, you should certainly open a ticket and we can dig deeper. Such tickets will always require thread dumps, multiples, across all cache servers. If you see that <13> stuck threads in this node message, the issue is likely snowballing and starting to impact this node, and the cluster could be next.

Gather artifacts, and take action. Perhaps a bounce of members, one at a time, for members showing stuck threads, would be prudent. Identifying which member to bounce can be difficult. That said, it is often possible, by analyzing the "15 seconds have elapsed" messages in your logs. This is described more in the Seconds have elapsed message in this document.

Disconnecting old DistributedSystem to prepare for a reconnect attempt

Attempting to reconnect to the DistributedSystem. This is attempt #n

Log Message:

```
[info 2021/09/21 22:45:37.863 EDT <ReconnectThread> tid=0x7f0d] Disconnecting old
DistributedSystem to prepare for a reconnect attempt
```

Log Level: info

Category: Membership

Meaning:

These messages are related, and may require action if you are not aware of why the member has been disconnected. This is often due to some instability in the distributed system caused by either network issues or GC related pauses.

Potential Resolutions:

Examine the logs of the member that is being forced out of the system. Perhaps the member became unresponsive. Look for other logging with keywords such as “elapsed”, “wakeup”, or “heartbeat”, all relatively unique words which can be searched for to proactively find potential issues. If any of these are discovered, GC tuning is likely needed.

Unable to form a TCP/IP connection in a reasonable amount of time

Log Message:

```
[info 2021/09/03 10:31:16.311 CDT <Timer-3> tid=0x79] Performing availability check
for suspect member aaa.bbb.ccc.ddd(member:5301)<v256>:41000 reason=Unable to form a
TCP/IP connection in a reasonable amount of time
```

Log Level: info, warning, fatal : Depending on the particular situation

Category: Membership

Meaning:

This message usually coincides with the availability check logging associated with suspect members. It should be investigated further by searching for other messages that may give more indication.

This specific message, if not accompanied by other “wakeup” or “heartbeat” messages, generally indicates that a member may have crashed unexpectedly, without warning. If, however, no member has crashed, the suspect member was able to respond during suspect processing and may no longer be at risk. Still, this definitely requires action to determine if you remain vulnerable to repeated occurrences.

Potential Resolutions:

This message alone doesn’t generally reveal how to proceed to eliminate issues. That said, a deep analysis of the logs for other significant related messages may be helpful, and following the potential resolutions for those could help to reduce or eliminate these messages.

Received Suspect Message

Log Message:

```
debug 2021/02/08 05:53:04.634 IST <member-43596> tid=0x2a] Suspecting member  XXX(serv
er1:40875)<v13>:41004
[info 2021/02/08 05:53:04.634 IST <member-43596> tid=0x2a] No longer suspecting
192.168.240.7(ch07node5:40875)<v13>:41004
[info 2021/03/29 06:46:56.304 EDT <Geode Failure Detection thread 162> tid=0x474f0c]
received suspect message from myself for XXX(YYY-server1:15972)<v16>:40000: SOME
REASON GENERALLY PROVIDED HERE
```

Log Level: info

Category: Membership

Meaning:

This message requires action. You are in danger of having a member kicked out of the distributed system, as it was already being “suspected” of being a problem for some unknown reasons that require investigation. Continuing to see these indicates that you are definitely not seeing optimal behavior or performance, and the system is thrashing with many messages thinking some member or members are unhealthy.

Potential Resolutions:

The “no longer suspecting” message is really an indication that the member is now considered healthy. However, it also means that the member was considered unhealthy and some member initiated “suspect” processing to determine if we should kick out the member to preserve the integrity and stability of the cluster. You will generally see suspect messages, shown above, for all members, as we send these out across the cluster to gather opinions. Ultimately, if the coordinator finds the member to be unresponsive within member-timeout seconds, the coordinator will kick out the member.

To take action, check the “Reason” seen in some of the logs, and take action accordingly. If this is rare, it is likely not an issue. If frequent, however, you definitely want to research and tune the system to eliminate these messages. If you are seeing the “no longer suspecting” message, that means that you should also see the “Suspecting member” message shown above. However, depending on your version of VMware GemFire, It may require debug level logging to see that message.

<n> Seconds Have Elapsed

Log Message:

```
[warn 2021/04/11 02:03:53.220 EDT <ServerConnection on port 10230 Thread 120>
tid=0xac97] 20 seconds have elapsed while waiting for replies:
<PRFunctionStreamingResultCollector 29058 waiting for 1 replies from [XXX]> on
YYY<v18>:10104 whose current membership list is: [LIST OF ALL MEMBERS]
[warn 2021/03/16 02:35:18.588 EDT <Timer-0> tid=0x2e] 15 seconds have elapsed waiting
for a response from XXX:14412)<v6>:40001 for thread ServerConnection on port 20102
Thread 592638
[warn 2021/04/15 03:30:59.107 EDT <main> tid=0x1] 15 seconds have elapsed while
waiting for replies: <DLockRequestProcessor 115 waiting for 1 replies from
[XXX(8582)-server2:68148)<v2>:41000]> on YYY<v2>:41001 whose current membership list
is: [LIST OF ALL MEMBERS]
```

Log Level: warning

Category: Membership

Meaning:

This message requires action. It is not necessarily urgent, but it is an indication that the messaging is taking much longer than expected between peers in your environment. The number of seconds displayed likely maps to the ack-wait-threshold in your environment, which defaults to 15 seconds. Some customers increase this setting, but it is recommended that you understand your environment first and only increase it if deemed necessary after attempting to correct any underlying causes of delays in communications.

Potential Resolutions:

This could be driven by GC related delays, JVM Pauses, burst of activity causing high peer to peer activity, threading issues, overwhelming CPU, etc. You could check for high replyWaitsInProgress across all nodes using JMX or stats analysis. If this is rare, it is not a likely cause for concern. If you are seeing this while experiencing high latency, it is likely an area to focus on. To analyze such issues, we will need all logs, stats, and gc logs across all members to identify which member or members is driving the slowdown.

NOTE: If many members have these messages, while another member does not appear to be waiting for replies from anybody, it is very likely that member is the source of the issue. After analysis to gather some information, you could try bouncing that member to see if this restores the other members to a healthier state.

Member isn’t responding to heartbeat requests

Log Message:

```
[info 2021/03/29 06:46:56.304 EDT <Geode Failure Detection thread 162> tid=0x474f0c]
received suspect message from myself for XXX(YYY-server1:15972)<v16>:40000: Member
isn't responding to heartbeat requests
[info 2021/02/21 00:38:33.436 GMT <unicast receiver,XXX-19638> tid=0x31] received
suspect message from YYY(cacheServer33010:16994)<v73>:33100 for
ZZZ(gcacheServer33010:27406)<v74>:33100: Member isn't responding to heartbeat
requests
[info 2021/02/21 00:38:33.440 GMT <Geode Failure Detection thread 10> tid=0x32f]
Performing availability check for suspect member
XXX(cacheServer33010:27406)<v74>:33100 reason=Member isn't responding to heartbeat
requests
```

Log Level: info

Category: Membership

Meaning:

This message requires **immediate** action. You are in danger of having a member kicked out of the distributed system due to being unresponsive. If the member continues to be unresponsive, the distributed system will kick out the member, to restore stability for the remaining members.

Potential Resolutions:

This is often related to a suboptimal heap and/or GC configuration. You could be experiencing JVM Pauses that require tuning. If you frequently see these messages without having the member kicked out, you have opportunities to tune and eliminate these messages. Alternatively, you could also increase the member-timeout property, however this is only suggested when you have full understanding of what is driving the member to be unresponsive to the heartbeat requests from the member monitoring it.

This message often corresponds with “suspect” messages, and members getting kicked out of the cluster. Logs, stats, and GC logs will be required in order to understand what is going on in this situation.

Enabled-network-partition-detection is set to false

Log Message:

```
[warning 2021/09/11 08:01:41.089 EDT locatorIS2 <Pooled Message Processor 1>
tid=0x48] Creating persistent region _ConfigurationRegion, but
enable-network-partition-detection is set to false. Running with network partition
detection disabled can lead to an unrecoverable system in the event of a network
split.
```

Log Level: warning

Category: Membership

Meaning:

This is a warning that you have chosen a configuration that makes you more susceptible to data consistency issues if you experience a network partition, or “split brain”. If you do choose this configuration and experience network issues that create a “split brain” scenario, where your distributed system splits into two separate distributed systems (DS), then it is possible that your data will diverge. Specifically, you could do puts into a region in DS A that do not make it into DS B, while also doing puts into DS B that do not make it into DS A. VMware GemFire will be unable to resolve this situation for you as you try to recover the system back into a single, healthy DS.

Potential Resolutions:

The best option is to choose to keep enable-network-partition-detection set to true. Beyond that, any split brain driven data divergence will require your manual intervention to avoid possible data loss.

Statistics sampling thread detected a wakeup delay

Log Message:

```
[warning 2021/02/09 21:37:44.728 EST member-49001 <StatSampler> tid=0x36] Statistics
sampling thread detected a wakeup delay of 40203 ms, indicating a possible resource
issue. Check the GC, memory, and CPU statistics.
```

Log Level: warning

Category: Membership

Meaning:

URGENT action is needed. You are experiencing JVM Pauses, where the JVM is preventing VMware GemFire from running at all for the given amount of time. This is only logged when the delay is at least 3 seconds more than your configured statistic-sample-rate. You are vulnerable to having members kicked out of the distributed system.

Potential Resolutions:

This is almost always caused by GC related behavior. To diagnose such issues, make sure to enable GC logging in your environment. If you have GC logs, search for “Full GC”, “concurrent mode failure”, “exhausted”, and other similar issues that drive long pauses. If you do open a ticket for assistance, please have VMware GemFire logs, stats, and GC logs ready to provide them prior to opening the ticket.

If this is urgent and you need immediate resolution without having time to fine tune GC, one possible temporary patch is to increase the member-timeout in the gemfire.properties file. This would make VMware GemFire more tolerant of processes being somewhat unresponsive for longer durations.

Redundancy has dropped below <n> configured copies

Log Message:

```
[warning 2021/03/23 09:26:51.641 EDT XXX-server01 <PartitionedRegion Message
Processor20> tid=0x1d66] Redundancy has dropped below 2 configured copies to 1 actual
copies for /RegionXYZ
[info 2021/03/23 09:26:51.798 EDT XXX-server01 <PartitionedRegion Message
Processor20> tid=0x1d66] Configured redundancy of 2 copies has been restored to
/RegionXYZ
```

Log Level: warning

Category: Membership

Meaning:

This message requires **immediate** action to determine if you are now vulnerable to data loss. This message indicates that you have lost access to 1 of the 2 configured copies of your data for that RegionXYZ on member XXX-server01. It is not necessarily urgent if you have redundancy configured and capacity for the remaining members to handle the increased load. The corresponding “has been restored” message, an info level message also shown above, indicates that you now are back to your healthy environment with redundancy in place for that RegionXYZ from the perspective of this member.

Potential Resolutions:

Investigate the cause of the loss in redundancy if it’s not already known. It could simply have been a planned maintenance that drove the cluster below configured redundancy levels. The settings that generally apply here are the number of copies configured, and then, the recovery-delay and startup-recovery-delay settings, which control whether and when we restore redundancy with the loss of a member of the distributed system and when it is added back in. Our documentation discusses these settings in detail.

Rejected connection

Log Message:

```
[warning 2021/05/10 12:28:31.063 BST gfcache.ivapp1237223.croydon.ms.com.7291
<Handshaker /0:0:0:0:0:0:0:0:7291 Thread 10> tid=0xa29] Rejected connection from XXX
because current connection count of 1,600 is greater than or equal to the configured
max of 1,600
[warn 2021/03/28 02:22:01.667 CDT <Handshaker 0.0.0.0/0.0.0.0:40404 Thread 23>
tid=0x85cf] Rejected connection from Server connection from [client host address=YYY;
client port=43198] because incoming request was rejected by pool possibly due to
thread exhaustion
```

Log Level: warning

Category: Communications

Meaning:

This message requires **URGENT** action. These messages indicate that you have exhausted resources, likely either due to using an insufficient max-connections setting for the cache server configuration or insufficient resources for the level of connection load on the system. Both of these messages are from the same area of code, trying to handle a new client connection request.

Potential Resolutions:

If you have increased load recently, or are using an old, legacy default value of 800 for max-connections, you may want to consider increasing this setting, regardless. Many customers use 2000, or even 5000 for those that do not want VMware GemFire to be throttling their performance/activity trying to conserve resources.

That said, if this number of connections is unexpected, you are potentially experiencing issues with connection timeouts, driving retry activity and a thrashing of resources that can cause the number of outstanding client connections and threads to be exhausted. You can observe this by examining VMware GemFire statistics using a tool like VSD, or, if using JMX, you can monitor usage with the CacheServeMXBean getClientConnectionCount() method. If you ever see unexpected spikes in this value, but are not seeing other symptoms, such as timeouts, perhaps you simply need to increase the max-connections appropriately.

However, if seeing these messages coincides with symptoms like client side timeouts, it could be due to an insufficient read-timeout in the client side pool configuration, or an insufficient accept queue on the server side. Another setting that warrants investigation is the BridgeServer.HANDSHAKE_POOL_SIZE. If you have not altered this setting in your system properties, you are likely using the default value of 4, which has been seen to be insufficient for many environments. Recommend increasing this VMware GemFire system property to at least 20.

PCC service metrics component failing to connect to locator/server

Log Message:

```
{"timestamp":"1620032802.654847383","source":"service-metrics","message":"service-metr
ics.executing-metrics-cmd","log_level":2,"data":{"error":"exit
status 1","event":"failed","output":"IOException error! MBeanServerConnection failed
to create.\njava.io.IOException: Failed to retrieve RMIServer stub:
javax.naming.ServiceUnavailableException [Root exception is
java.rmi.ConnectException: Connection refused to host:
461737ba-07ca-4897-9e41-a70ae7f26274.server.services.service-instance-94fbf6cc-4073-4a
45-8965-7ea855bcd0ca.bosh;
nested exception is: \n\tjava.net.ConnectException: Connection refused (Connection
refused)]\nException in thread \"main\" java.lang.NullPointerException\n\tat
io.pivotal.cloudcache.metrics.cli.MetricsExtractor.lambda$static$0(MetricsExtractor.ja
va:10)\n\tat
io.pivotal.cloudcache.metrics.cli.JMXPropertiesEmitter.lambda$getMemberMetrics$0(JMXPr
opertiesEmitter.java:55)\n\tat
java.util.stream.ReferencePipeline$3$1.accept(ReferencePipeline.java:193)\n\tat
java.util.Spliterators$ArraySpliterator.forEachRemaining(Spliterators.java:948)\n\tat
java.util.stream.AbstractPipeline.copyInto(AbstractPipeline.java:482)\n\tat
java.util.stream.AbstractPipeline.wrapAndCopyInto(AbstractPipeline.java:472)\n\tat
java.util.stream.ReduceOps$ReduceOp.evaluateSequential(ReduceOps.java:708)\n\tat
java.util.stream.AbstractPipeline.evaluate(AbstractPipeline.java:234)\n\tat
java.util.stream.ReferencePipeline.collect(ReferencePipeline.java:566)\n\tat
io.pivotal.cloudcache.metrics.cli.JMXPropertiesEmitter.getMemberMetrics(JMXPropertiesE
mitter.java:56)\n\tat
io.pivotal.cloudcache.metrics.cli.JMXPropertiesEmitter.main(JMXPropertiesEmitter.java:
30)\n"}}
{"timestamp":"1620032842.481263161","source":"service-metrics","message":"service-metr
ics.executing-metrics-cmd","log_level":1,"data":{"event":"starting"}}
```

Category: Communications

Meaning:

Every VM in PCC for locators or servers has its own service-metrics component. The job of this component is to periodically check the health of the VMware GemFire server/locator processes running. The way it does that job is by making an RMI call to the JMX manager. When it cannot connect to the locator/server process, it starts logging these errors in its own log.

SSLHandshakeException: <version> is disabled

Log Message:

```
[warn 2021/04/26 15:44:52.418 EDT kbc000100.rw.example.com <P2P message reader@388969b
8> tid=0x3a] SSL handshake exception
javax.net.ssl.SSLHandshakeException: <<ssl_version>> is disabled
        at sun.security.ssl.InputRecord.handleUnknownRecord(InputRecord.java:637)
        at sun.security.ssl.InputRecord.read(InputRecord.java:527)
        at sun.security.ssl.EngineInputRecord.read(EngineInputRecord.java:382)
        at sun.security.ssl.SSLEngineImpl.readRecord(SSLEngineImpl.java:951)
        at sun.security.ssl.SSLEngineImpl.readNetRecord(SSLEngineImpl.java:896)
        at sun.security.ssl.SSLEngineImpl.unwrap(SSLEngineImpl.java:766)
        at javax.net.ssl.SSLEngine.unwrap(SSLEngine.java:624)
        at org.apache.geode.internal.net.NioSslEngine.handshake(NioSslEngine.java:148)
        at org.apache.geode.internal.net.SocketCreator.handshakeSSLSocketChannel(Socke
tCreator.java:840)
        at org.apache.geode.internal.tcp.Connection.createIoFilter(Connection.java:174
7)
        at org.apache.geode.internal.tcp.Connection.readMessages(Connection.java:1548)
        at org.apache.geode.internal.tcp.Connection.run(Connection.java:1472)
        at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1
149)
        at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:
624)
        at java.lang.Thread.run(Thread.java:748)
```

Category: Communications

Meaning:

This means the specified SSL/TLS protocol is not compatible with, or configured correctly, on one or more members. The simplest workaround is to use “any'' as the protocol, however, some customers have strict security requirements that mandate specific versions and ciphers, which will require that all members are configured with compatible (matching) protocols and ciphers and that those protocols/ciphers are supported by the underlying JRE.

Unable To Create New Native Thread

Log Message:

```
java.lang.OutOfMemoryError: unable to create new native thread
```

Log Level: warning

Category: Heap / JVM / GC

Meaning:

The JVM needs various resources to create a new 'native' thread, which may not map one-to-one with application threads. These resources are external to the JVM heap and include "native" memory for the stack and, potentially, user processes.

Potential Resolution:

Depending on the resource limit encountered, you may need to increase the maximum number of user processes as configured with ulimit and/or "/etc/security/limits.conf", or you may not have sufficient system memory. In the latter case, you will need to make more system memory available and/or decrease the amount of stack memory used per thread. If you have excess, unused heap under even heavy load, you may be able to reduce the heap size and leave more memory for "native" usage.

Alternatively, you might be able to decrease the stack size of each thread, by setting the JVM parameter "-xss" to something smaller (the defaults are 320 KB for 32-bit JVMs and 1024 KB for 64-bit JVMs), but this must be done with care as it can cause threads to not have enough stack to properly operate. The last and safest option is to add free memory to the system by either adding memory or reducing other consumers of system memory (e.g. other applications).

Too Many Open Files

Log Message:

```
java.net.SocketException: Too many open files (Socket creation failed/Accept failed)
```

Log Level: warning

Category: Communications

Meaning:

The number of sockets available to your applications is governed by operating system limits. Sockets use file descriptors and the operating system's view of your application's socket use is expressed in terms of file descriptors.

Potential Resolution:

There are two limits on the maximum descriptors available to a single application, a soft limit, which can be increased using the ulimit command as a user, and a "hard" limit which will require editing "/etc/security/limits.conf" and relogging in. (There is also an OS level limit that will require a system administrator to tune kernel parameters, however, this limit is typically large and is rarely hit.) It is also possible that the FD's being consumed are being driven by a major increase of connections/threads due to some burst of activity or connections timing out. This can lead to retry activity driving the number of open files to increase. If you increase the soft and hard limits, and continue to observe these messages, you may need to analyze whether you have connections timing out, sufficient TCP accept queue, etc. This can require an increase of the p2p.backlog and net.core.somaxconn settings.

CommitConflictException

Log Message:

```
org.apache.geode.cache.CommitConflictException: Concurrent transaction commit
detected The key xxx in region /xxx was being modified by another transaction locally
```

Log Level: warning

Category: Operations

Meaning:

You design transactions such that any get operations are within the transaction. This causes those entries to be part of the transactional state, which is desired such that intersecting transactions can be detected and signal commit conflicts. You have to catch the commit conflicts in your code like this:

```
 try {
  txmgr.begin();
   // add your codes here
  txmgr.commit();
   }
 catch (CommitConflictException conflict) {
  // entry value changed causing a conflict, so try again
  } finally {
  //add your codes here
  }
 }
```

Initialization of Region <_B__RegionName_BucketNumber> Completed

Log Message:

```
[info 2021/03/28 00:41:20.047 EDT <Recovery thread for bucket _B__RegionName_32>
tid=0x164] Region _B__RegionName_32 requesting initial image from
IP(gemfire-server-1:88590)<v19>:10100

[info 2021/03/28 00:41:20.048 EDT <Recovery thread for bucket _B__RegionName_32>
tid=0x164] _B__RegionName_32 is done getting image from
IP(gemfire-server-1:88590)<v19>:10100. isDeltaGII is true

[info 2021/03/28 00:41:20.048 EDT <Recovery thread for bucket
 _B__firm_0_RegionName_32> tid=0x164] Region _B__RegionName_32 initialized persistent
 id: /IP:/pathTo-server-1/cldist created at timestamp 1616906479201 version 0
 diskStoreId DiskStoreid name null with data from
 IP(gemfire-server-1:88590)<v19>:10100.

[info 2021/03/28 00:41:20.048 EDT <Recovery thread for bucket _B__RegionName_32>
tid=0x164] Could not initiate event tracker from GII provider
IP(gemfire-server-1:88590)<v19>:10100
```

```
[info 2021/03/28 00:41:20.048 EDT <Recovery thread for bucket _B__RegionName_32>
tid=0x164] Initialization of region _B__RegionName_32 completed
```

Log Level: info

Category: Membership

Meaning:

This set of messages are related to the initialization of Partitioned regions. They indicate where the VMware GemFire system is retrieving each bucket from to perform this initialization. In the above example, bucket 32 for region "RegionName" is being retrieved from member gemfire-server-1 as VMware GemFire believes this to be the most recent data for that bucket. This is the "requesting initial image" message above. The "Initialization of region <> completed message can be useful to determine where each specific bucket, for each specific region, is located across the membership.

Potential Resolution:

There is no "resolution" here, but customers have asked how to determine where each bucket exists across the cluster. Using the above message can be very useful to filter the logs to see exactly where each bucket exists in the cluster, for each region. One could use a command such as one like this: `egrep -R --include=\*.log 'Initialization of region _B__RegionName_' ~/PathToLogFiles//gflogs/*`. The above command could tell you exactly where each bucket exists for region RegionName. If you use only `Initialization of region _B__` instead, this would then output the buckets across all partitioned regions. This output could then be used to know where each specific bucket exists across the cluster, to serve whatever purpose you deem helpful in monitoring your cluster. There does exist some great documentation and project for how to identify where buckets are located in this article: https://community.pivotal.io/s/article/GemFire-Monitoring-PR-Entry-and-Bucket-Details.

Unknown pdx Type error

Log Message:

```
Caused by: java.lang.IllegalStateException: Unknown pdx type=X
at com.gemstone.gemfire.internal.InternalDataSerializer.readPdxSerializable(InternalDa
taSerializer.java:2977)
at com.gemstone.gemfire.internal.InternalDataSerializer.basicReadObject(InternalDataSe
rializer.java:2794)
at com.gemstone.gemfire.DataSerializer.readObject(DataSerializer.java:3212)
at com.gemstone.gemfire.internal.util.BlobHelper.deserializeBlob(BlobHelper.java:81)
at com.gemstone.gemfire.internal.cache.EntryEventImpl.deserialize(EntryEventImpl.java:
1407)
at com.gemstone.gemfire.internal.cache.PreferBytesCachedDeserializable.getDeserialized
Value(PreferBytesCachedDeserializable.java:65)
at com.gemstone.gemfire.cache.query.internal.index.DummyQRegion.getValues(DummyQRegio
n.java:153)
at com.gemstone.gemfire.cache.query.internal.index.DummyQRegion.values(DummyQRegion.ja
va:109)
at com.gemstone.gemfire.cache.query.internal.index.DummyQRegion.iterator(DummyQRegion.
java:198)
at com.gemstone.gemfire.cache.query.internal.index.CompactRangeIndex$IMQEvaluator.doNe
stedIterations(CompactRangeIndex.java:1763)
at com.gemstone.gemfire.cache.query.internal.index.CompactRangeIndex$IMQEvaluator.eval
```

```
uate(CompactRangeIndex.java:1622)
... 27 more
```

Log Level: Error

Category: Operations

Meaning:

A Portable Data eXchange (PDX) related exception that may occur when restarting a distributed system without also restarting any clients. When using PDX serialization without persistence, the above exception may be seen on a client after bouncing all of the servers of the distributed system without restarting the client. Generally, this message indicates that the PDX metadata on some client is out-of-sync with the servers.

Potential Resolution:

To avoid this issue without persisting PDX types on server members, you must restart your client application when restarting the servers. Alternately, to avoid this issue without restarting your client application, you must enable PDX persistence on servers. By doing this, you are guaranteed that any already defined PDX types will remain available between server restarts. This doesn't require storing the data from your regions, you can store only PDX metadata, regions data, or both. Below mentioned is an example of how to configure PDX persistence on the server side:

```
<disk-store name="pdxDiskStore">
<disk-dirs>
<disk-dir>pdxDiskStore</disk-dir>
</disk-dirs>
</disk-store>

<pdx read-serialized="true" persistent="true" disk-store-name="pdxDiskStore"/>
```

Error calculating expiration

Log Message:

```
2021-06-02 12:35:26,071 FATAL o.a.g.i.c.LocalRegion [Recovery thread for bucket _B__gd
c__eventsLow_50] Error calculating expiration An IOException was thrown while deserial
izing
org.apache.geode.SerializationException: An IOException was thrown while deserializing
        at org.apache.geode.internal.cache.EntryEventImpl.deserialize(EntryEventImpl.j
ava:2041) ~[geode-core-9.10.5.jar:?]
        at org.apache.geode.internal.cache.EntryEventImpl.deserialize(EntryEventImpl.j
ava:2032) ~[geode-core-9.10.5.jar:?]
        at org.apache.geode.internal.cache.VMCachedDeserializable.getDeserializedValue
(VMCachedDeserializable.java:113) ~[geode-core-9.10.5.jar:?]
        at org.apache.geode.internal.cache.LocalRegion.getDeserialized(LocalRegion.jav
a:1280) ~[geode-core-9.10.5.jar:?]
        at org.apache.geode.internal.cache.ExpiryRegionEntry.getValue(ExpiryRegionEntr
y.java:101) ~[geode-core-9.10.5.jar:?]
        at com.ihg.enterprise.gdc.model.CustomExpiryHandler.getExpiry(CustomExpiryHand
ler.java:19) ~[gdc-gemfire-side-2.26-jar-with-dependencies.jar:?]
        at org.apache.geode.internal.cache.LocalRegion.createExpiryTask(LocalRegion.ja
va:7774) ~[geode-core-9.10.5.jar:?]
        at org.apache.geode.internal.cache.LocalRegion.addExpiryTask(LocalRegion.java:
7901) ~[geode-core-9.10.5.jar:?]
```

```
        at org.apache.geode.internal.cache.LocalRegion.addExpiryTask(LocalRegion.java:
7753) ~[geode-core-9.10.5.jar:?]
        at org.apache.geode.internal.cache.LocalRegion.lambda$rescheduleEntryExpiryTas
ks$3(LocalRegion.java:7741) ~[geode-core-9.10.5.jar:?]
        at org.apache.geode.internal.cache.ExpiryTask.doWithNowSet(ExpiryTask.java:48
0) [geode-core-9.10.5.jar:?]
        at org.apache.geode.internal.cache.LocalRegion.rescheduleEntryExpiryTasks(Loca
lRegion.java:7739) [geode-core-9.10.5.jar:?]
        at org.apache.geode.internal.cache.LocalRegion.initialize(LocalRegion.java:239
4) [geode-core-9.10.5.jar:?]
        at org.apache.geode.internal.cache.DistributedRegion.initialize(DistributedReg
ion.java:1099) [geode-core-9.10.5.jar:?]
        at org.apache.geode.internal.cache.BucketRegion.initialize(BucketRegion.java:2
59) [geode-core-9.10.5.jar:?]
        at org.apache.geode.internal.cache.LocalRegion.createSubregion(LocalRegion.jav
a:983) [geode-core-9.10.5.jar:?]
        at org.apache.geode.internal.cache.PartitionedRegionDataStore.createBucketRegi
on(PartitionedRegionDataStore.java:784) [geode-core-9.10.5.jar:?]
        at org.apache.geode.internal.cache.PartitionedRegionDataStore.grabFreeBucket(P
artitionedRegionDataStore.java:459) [geode-core-9.10.5.jar:?]
        at org.apache.geode.internal.cache.PartitionedRegionDataStore.grabBucket(Parti
tionedRegionDataStore.java:2875) [geode-core-9.10.5.jar:?]
        at org.apache.geode.internal.cache.ProxyBucketRegion.recoverFromDisk(ProxyBuck
etRegion.java:463) [geode-core-9.10.5.jar:?]
        at org.apache.geode.internal.cache.ProxyBucketRegion.recoverFromDiskRecursivel
y(ProxyBucketRegion.java:406) [geode-core-9.10.5.jar:?]
        at org.apache.geode.internal.cache.PRHARedundancyProvider$2.run2(PRHARedundanc
yProvider.java:1640) [geode-core-9.10.5.jar:?]
        at org.apache.geode.internal.cache.partitioned.RecoveryRunnable.run(RecoveryRu
nnable.java:60) [geode-core-9.10.5.jar:?]
        at org.apache.geode.internal.cache.PRHARedundancyProvider$2.run(PRHARedundancy
Provider.java:1630) [geode-core-9.10.5.jar:?]
        at java.lang.Thread.run(Thread.java:748) [?:1.8.0_212]
Caused by: java.io.IOException: Unknown header byte 83
        at org.apache.geode.internal.serialization.DscodeHelper.toDSCODE(DscodeHelper.
java:40) ~[geode-serialization-9.10.5.jar:?]
        at org.apache.geode.internal.InternalDataSerializer.basicReadObject(InternalDa
taSerializer.java:2494) ~[geode-core-9.10.5.jar:?]
        at org.apache.geode.DataSerializer.readObject(DataSerializer.java:2864) ~[geod
e-core-9.10.5.jar:?]
        at org.apache.geode.internal.util.BlobHelper.deserializeBlob(BlobHelper.java:9
9) ~[geode-core-9.10.5.jar:?]
        at org.apache.geode.internal.cache.EntryEventImpl.deserialize(EntryEventImpl.j
ava:2039) ~[geode-core-9.10.5.jar:?]
        ... 24 more
```

Log Level: Warning

Category: Storage

Potential Resolution:

This is due to inconsistencies in the data stored on region/disk vs. the PdxType and may throw during deserialization. Cleaning the data or syncing it according to PdxType is a possible solution.

PdxType limitations for GFSH queries

Log Message:

```
[info 2021/06/15 13:01:24.238 EDT 170834GFCluster.sd-1d7e-bd1c.cacheServer40404 <Funct
ion Execution Processor3> tid=0x7b] Exception occurred:
org.apache.geode.pdx.JSONFormatterException: Could not create JSON document from PdxIn
stance
        at org.apache.geode.pdx.JSONFormatter.toJSON(JSONFormatter.java:173)
        at org.apache.geode.management.internal.cli.domain.DataCommandResult$SelectRes
ultRow.valueToJson(DataCommandResult.java:726)
        at org.apache.geode.management.internal.cli.domain.DataCommandResult$SelectRes
ultRow.resolveStructToColumns(DataCommandResult.java:712)
        at org.apache.geode.management.internal.cli.domain.DataCommandResult$SelectRes
ultRow.resolveObjectToColumns(DataCommandResult.java:689)
        at org.apache.geode.management.internal.cli.domain.DataCommandResult$SelectRes
ultRow.createColumnValues(DataCommandResult.java:679)
        at org.apache.geode.management.internal.cli.domain.DataCommandResult$SelectRes
ultRow.<init>(DataCommandResult.java:662)
        at org.apache.geode.management.internal.cli.functions.DataCommandFunction.crea
teSelectResultRow(DataCommandFunction.java:266)
        at org.apache.geode.management.internal.cli.functions.DataCommandFunction.sele
ct_SelectResults(DataCommandFunction.java:252)
        at org.apache.geode.management.internal.cli.functions.DataCommandFunction.sele
ct(DataCommandFunction.java:220)
        at org.apache.geode.management.internal.cli.functions.DataCommandFunction.sele
ct(DataCommandFunction.java:173)
        at org.apache.geode.management.internal.cli.functions.DataCommandFunction.exec
ute(DataCommandFunction.java:122)
        at org.apache.geode.internal.cache.MemberFunctionStreamingMessage.process(Memb
erFunctionStreamingMessage.java:193)
        at org.apache.geode.distributed.internal.DistributionMessage.scheduleAction(Di
stributionMessage.java:367)
        at org.apache.geode.distributed.internal.DistributionMessage$1.run(Distributio
nMessage.java:430)
        at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1
149)
        at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java)
        at org.apache.geode.distributed.internal.ClusterDistributionManager.runUntilSh
utdown(ClusterDistributionManager.java:952)
        at org.apache.geode.distributed.internal.ClusterDistributionManager.doFunction
ExecutionThread(ClusterDistributionManager.java:806)
        at org.apache.geode.internal.logging.LoggingThreadFactory.lambda$newThread$0(L
oggingThreadFactory.java:121)
        at java.lang.Thread.run(Thread.java:748)
Caused by: java.lang.IllegalStateException: PdxInstance returns unknwon pdxfield value
for type Wed Apr 07 00:00:00 EDT 2021
        at org.apache.geode.pdx.internal.json.PdxToJSON.writeValue(PdxToJSON.java:144)
        at org.apache.geode.pdx.internal.json.PdxToJSON.getJSONString(PdxToJSON.java:1
75)
        at org.apache.geode.pdx.internal.json.PdxToJSON.writeValue(PdxToJSON.java:138)
        at org.apache.geode.pdx.internal.json.PdxToJSON.getJSONString(PdxToJSON.java:1
75)
        at org.apache.geode.pdx.internal.json.PdxToJSON.writeValue(PdxToJSON.java:138)
        at org.apache.geode.pdx.internal.json.PdxToJSON.getJSONString(PdxToJSON.java:1
75)
        at org.apache.geode.pdx.internal.json.PdxToJSON.writeValue(PdxToJSON.java:138)
        at org.apache.geode.pdx.internal.json.PdxToJSON.getJSONString(PdxToJSON.java:1
75)
        at org.apache.geode.pdx.internal.json.PdxToJSON.writeValue(PdxToJSON.java:138)
        at org.apache.geode.pdx.internal.json.PdxToJSON.getJSONString(PdxToJSON.java:1
75)
        at org.apache.geode.pdx.internal.json.PdxToJSON.getJSON(PdxToJSON.java:57)
        at org.apache.geode.pdx.JSONFormatter.toJSON(JSONFormatter.java:171)
        ... 19 more
```

Log Level: INFO

Category: Operations

Potential Resolution:

Other than primitive types like object types (String, Character, Date etc.) will not be deserialized on GFSH queries.

Apache.Geode.Client.AllConnectionsInUseException

Log Message:

In StdOut/StdError on Client Side:

```
Apache.Geode.Client.AllConnectionsInUseException
Region::getAll: All connections are in use
Apache.Geode.Client.Region`2[[System.__Canon, mscorlib],[System.__Canon, mscorlib]].Ge
tAll(System.Collections.Generic.ICollection`1<System.__Canon>, System.Collections.Gene
ric.IDictionary`2<System.__Canon,System.__Canon>, System.Collections.Generic.IDictiona
ry`2<System.__Canon,System.Exception>, Boolean)
   Apache.Geode.Client.Region`2[[System.__Canon, mscorlib],[System.__Canon, mscorli
b]].GetAll(System.Collections.Generic.ICollection`1<System.__Canon>, System.Collection
s.Generic.IDictionary`2<System.__Canon,System.__Canon>, System.Collections.Generic.IDi
ctionary`2<System.__Canon,System.Exception>)
```

Category: Operations

Meaning:

This is evidence of the connection pool getting overwhelmed on the client side and not a problem on the VMware GemFire server side. Resolution: Increase the max-connections property to higher value as appropriate on pool settings on native client.

org.apache.geode.pdx.PdxInitializationException

Log Message: / Stack-trace / StdError:

```
The Cache Server process terminated unexpectedly with exit status 1. Please refer to t
he log file in /appdata/gemfire/edl/data/server for full details.
Exception in thread "main" org.apache.geode.pdx.PdxInitializationException: Could not
create pdx registry
    at org.apache.geode.pdx.internal.PeerTypeRegistration.initialize(PeerTypeRegistrat
ion.java:204)
    at org.apache.geode.pdx.internal.TypeRegistry.creatingDiskStore(TypeRegistry.java:
267)
    at org.apache.geode.internal.cache.DiskStoreFactoryImpl.create(DiskStoreFactoryImp
l.java:160)
    at org.apache.geode.internal.cache.xmlcache.CacheCreation.createDiskStore(CacheCre
ation.java:792)
    at org.apache.geode.internal.cache.xmlcache.CacheCreation.initializePdxDiskStore(C
acheCreation.java:783)
    at org.apache.geode.internal.cache.xmlcache.CacheCreation.create(CacheCreation.jav
a:507)
```

```
    at org.apache.geode.internal.cache.xmlcache.CacheXmlParser.create(CacheXmlParser.j
ava:338)
    at org.apache.geode.internal.cache.GemFireCacheImpl.loadCacheXml(GemFireCacheImpl.
java:4294)
```

Category: Operations

Meaning:

This explains that the PdxRegion is not getting initialized due to the corrupted cluster configs.

Potential Resolution:

Stop locator(s), then clear the cluster configs/pdx disk stores and, finally, start the locator(s). KB exists: https://community.pivotal.io/s/article/Fails-to-Start-a-Cache-Member-with-orgapachegeodepdxPdxInitializationException-Could-not-create-pdx-registry?language=en_US.

Format of the string <<cache xml file's content>> used for parameterization is unresolvable

Note: the spelling "perameterization" is wrong in the codebase https://github.com/apache/geode/blob/a5bd36f9fa787d3a71c6e6efafed5a7b0fe52d2b/geode-core/src/main/java/org/apache/geode/internal/cache/xmlcache/CacheXmlPropertyResolver.java#L125. Working to report & fix this.

Log Message:

```
[error 2021/09/08 11:42:16.830 EDT <main> tid=0x1] Format of the string <?xml version
="1.0"?>
<cache xmlns="http://geode.apache.org/schema/cache"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://geode.apache.org/schema/cache http://geode.apache.org/s
chema/cache/cache-1.0.xsd"
    version="1.0">
<gateway-sender id="GW_SENDER_${REMOTE_DS_ID_1}" remote-distributed-system-id="${REMOT
E_DS_ID_1}" manual-start="${GW_START_ID_1}" parallel="false" enable-persistence="true"
disk-store-name="gw1" disk-synchronous="true" dispatcher-threads="5" socket-read-timeo
ut="300000"/>
<gateway-sender id="GW_SENDER_${REMOTE_DS_ID_2}" remote-distributed-system-id="${REMOT
E_DS_ID_2}" manual-start="${GW_START_ID_2}" parallel="false" enable-persistence="true"
disk-store-name="gw2" disk-synchronous="true" dispatcher-threads="5" socket-read-timeo
ut="300000"/>
  <gateway-receiver hostname-for-senders="${HOSTNAME_FOR_SENDERS}" bind-address="${HOS
TNAME}" start-port="1531" end-port="1532"/>
-
-
-
</cache>
 used for parameterization is unresolvable


[error 2021/09/08 11:42:17.260 EDT <main> tid=0x1] Cache initialization for GemFireCac
he[id = -29791992; isClosing = false; isShutDownAll = false; created = Wed Sep 08 11:3
9:37 EDT 2021; server = false; copyOnRead = false; lockLease = 120; lockTimeout = 60]
failed because: java.lang.NullPointerException
```

Category: Configuration

Meaning:

This error occurs when the parameterized values are provided for the properties/attributes. For example, `manual-start="${GW_START_ID_1}"`. This property expects boolean, but seems to be getting some non-boolean value.

Potential Resolution:

Fix the errors for incorrect values or their types when provided dynamically.

RegionExistException

Log Message:

```
[error 2021/09/29 11:25:48.885 CDT <main> tid=0x1] Cache initialization for
GemFireCache[id = 755944228; isClosing = false; isShutDownAll = false; created = Wed
Sep 29 11:25:46 CDT 2021; server = false; copyOnRead = false; lockLease = 120;
lockTimeout = 60] failed because: org.apache.geode.cache.CacheXmlException: While
parsing XML, caused by org.xml.sax.SAX
Exception: A CacheException was thrown while parsing XML.
org.apache.geode.cache.RegionExistsException: /RegionX
```

Log Level: Error

Category: Configuration

Meaning:

This message indicates that the locator already has region (RegionX) in the cluster configuration and while starting up, a duplicate region is being created via API or cache.xml.

Potential Resolutions:

Remove duplicate region definition from the configurations.

- If “enable-cluster-configuration=true” in locator properties, then do the following:
  - Export the cluster configuration (`export cluster-configuration --xml-file=value`)
  - Remove the duplicate Region definition
  - Re-import the cluster configuration (`import cluster-configuration --action=STAGE`) and restart.
- If “enable-cluster-configuration=false” in locator properties, then remove the duplicate region definition from cache.xml.

Missing Diskstore Exception

Log Message:

```
Region /RegionX has potentially stale data.
It is waiting for another member to recover the latest data.
My persistent id:


  DiskStore ID: 6893751ee74d4fbd-b4780d844e6d5ce7
  Name: server1
  Location: /192.0.2.0:/home/grawat/server1/.


Members with potentially new data:
[
  DiskStore ID: 160d415538c44ab0-9f7d97bae0a2f8de
  Name: server2
  Location: /192.0.2.0:/home/grawat/server2/.
]
Use the "gfsh show missing-disk-stores" command to see all disk stores
that are being waited on by other members.
```

Log Level: Info

Category: Storage

Meaning:

When you start a member with a persistent region, the data is retrieved from disk stores to recreate the member's persistent region. If the member does not hold all of the most recent data for the region, then other members have the data, and region creation blocks, waiting for those other members. A partitioned region with colocated entries also blocks on start up, waiting for the entries of the colocated region to be available. So, this message shows that the disk store for server2 has the most recent data for the region, and server1 is waiting for server2.

Potential Resolutions:

- Start all members with persisted data first and at the same time.
- Respond to the waiting members by starting the server on which the waiting member is waiting on.

Could not create an instance of a class

Log Message:

```
Could not create an instance of a class com.xxx.yyy.zzz
```

Log Level: Error

Category: Configuration

Meaning:

This message indicates that either the class is not available in the classpath or the jar, which contains this class, is not deployed on Cache servers.

Potential Resolutions:

- Make the class available to classpath.
- Deploy the class on cache servers.

PartitionedRegion#cleanupFailedInitialization: Failed to clean the PartitionRegion allPartitionedRegions

Log Message:

```
[warning 2021/05/15 08:51:46.460 EDT 170834GFCluster.lmwcpbacap01p.cacheServer50506
<main> tid=0x1] PartitionedRegion#cleanupFailedInitialization: Failed to clean the
PartionRegion allPartitionedRegions
org.apache.geode.distributed.DistributedSystemDisconnectedException: Distribution
manager on 10.102.8.41(cacheServer50506:278621)<v1>:50001 started at Sat May 15
08:44:31 EDT 2021: Failed to acknowledge a new membership view and then failed tcp/ip
connection attempt, caused by org.apache.geode.ForcedDisconnectException: Failed to
acknowledge a new membership view and then failed tcp/ip connection attempt
```

Log Level: Error

Category: Membership

Meaning:

This message indicates that the buckets for partitioned regions have not recovered fully but a destroy region is issued for the regions whose buckets are still recovering.

Potential Resolutions:

Make sure that regions are recovered before issuing any destroy command.

Could not find any server to create primary client queue on.

Log Message:

```
[error 2016/09/13 10:45:29.351 PDT client tid=0x34] Could not find any server to
create primary client queue on. Number of excluded servers is 0 and the exception is
null.
```

Log Level: Error

Category: Communications

Meaning:

When a client with subscription-enabled="true" is started, messages like below will be logged in the VMware GemFire client log. If subscription-redundancy is not set, there will be one of these; if it is set to 1, there will be two, etc. The Cache Client Updater Thread is the thread waiting for events from the server. If no other server is available to which the Cache Client Updater Thread is connected, then above error message will be logged:

Potential Resolutions:

Make sure that the server, to which the Cache Client Updater Thread is connected, is up and running.

Cluster configuration service not available

Log Message:

```
Exception in thread "main" org.apache.geode.GemFireConfigException: cluster configurat
ion service not available
  at org.apache.geode.internal.cache.GemFireCacheImpl.requestSharedConfiguration(GemFi
reCacheImpl.java:1265)
```

Log Level: Error

Category: Configuration

Meaning:

This message indicates that the cache server is configured with, “use-cluster-configuration = true”, but is unable to get the cluster configuration from the locator.

Potential Resolutions:

Ensure that the locator has “enable-cluster-configuration=true” and the cache servers are able to get the cluster configurations from locators.

The secondary map already contained an event from hub null so ignoring new event

Log Message:

```
[warn 2021/09/10 15:49:31.692 CEST <P2P message reader for
00.01.02.03(some-node:17718)<v2>:41001 unshared ordered uid=1086 dom #1 port=39166>
tid=0x247bb] AsyncEventQueue_SubstitutionEventQ: The secondary map already contained
an event from hub null so ignoring new event GatewaySenderEventImpl[id=EventID[id=25
bytes...
```

Log Level: Warn

Category: Operations

Meaning:

This message indicates that the secondary gateway sender, hosted by a server, received an event that was already processed by the primary gateway sender of other server B; so the event itself shouldn’t be added to the internal map of unprocessed events.

Potential Resolutions:

This message, if seen occasionally, is harmless in most situations.

Create is present in more than one Oplog. This should not be possible. The Oplog Key ID for this entry is

Log Message:

```
java.lang.AssertionError: Oplog::readNewEntry: Create is present in more than one Oplo
g. This should not be possible
```

Log Level: Error

Category: Storage

Meaning:

This error indicates that the oplog is corrupted which makes it harder to write/delete any new entry in the oplogs (Diskstores) because of which cache servers will have trouble starting.

Potential Resolutions:

Clean up the disk stores.

Detected conflicting PDX types during import

Log Message:

```
Could not process command due to error. Detected conflicting PDX types during import
```

Log Level: Error

Category: Operations

Meaning:

When data is imported into a cluster with pdx serialization with existing data using gfsh import/export command and if the receiving cluster already has some data with different pdx metadata, the import will fail with the error.

Potential Resolutions:

Import data in the empty cluster or programmatically read the .gfd file and then perform the put operation.

A tenured heap garbage collection has occurred

Log Message:

```
[info 2021/10/13 17:14:56.177 EDT memberXXX <Notification Handler1> tid=0x69] A
tenured heap garbage collection has occurred.  New tenured heap consumption:
492250720
```

Log Level: Info

Category: Heap/GC/JVM/OS

Meaning:

This message occurs when a tenured space garbage collection has occurred. The goal is to provide the customer with a very accurate read for how much heap is actually consumed. External monitors do not know when a collection has occurred. The value specified is how much live data exists in tenured heap.

If you see this value constantly increasing over time, without a similar rate of increase of VMware GemFire entries, then this warrants some investigation into potential leaks. Short term increases due to queries, for example, are not worthy of concern, other than providing an indication that finer tuning may be warranted. The short term data resulting from a query would hopefully be fulfilled using the young generation heap, most of the time.

Potential Resolutions:

No resolution necessary. This is informative only. If you see this message frequently, however, it is a sign that you may need more heap, or finer tuning. You may be imbalanced unknowingly, etc. If seeing this message more frequently than every 1 hour, consistently, it is a sign that you may need tuning. Note: G1GC “mixed” collections may not drive this message, unless you are using more current versions of the JDK.

Allocating larger network read buffer

Log Message:

```
[info 2021/10/13 17:14:56.177 EDT locator <P2P message reader for
192.168.1.5(server1:8438)<v1>:41001 shared unordered sender uid=1 local port=42181
remote port=57345> tid=0x4c] Allocating larger network read buffer, new size is
134074 old size was 32768.
```

Log Level: Info

Category: Communications

Meaning:

This may require configuration change, to give more optimal behavior. If you have different socket-buffer-sizes across the various members of your distributed system, including locators, this message may be a sign that messages are potentially being lost. This can lead to distributed deadlocks. This message essentially means that the system is needing to grow and shrink to handle the messaging between the members.

Potential Resolutions:

Set all members, including locators, to the same socket-buffer-size. If you have seen this message, and appear to be impacted in the system, it may warrant some deeper analysis of the health of the system. Check for stuck threads, potentially gather thread dumps, to assess whether you are impacted.

Socket send buffer size is <m> instead of the requested <n>

Log Message:

```
[info 2021/11/19 13:51:47.569 PST server1 <P2P message reader@75099de0> tid=0x30]
Socket send buffer size is 6710884 instead of the requested 16777215.
```

Log Level: Info

Category: Communications

Meaning:

This may require configuration change, to give more optimal behavior. This message tells you that your VMware GemFire configuration is specifying a larger socket-buffer-size that the lower OS is going to permit. Hence, you see this message, and perhaps less than optimal behavior.

Potential Resolutions:

Make sure to set all members OS configurations to be the same, similar enough to avoid having this less than optimal potential chunking of messages when sending messages between members of the VMware GemFire distributed system.

quorum has been lost

Log Message:

```
[warn 2021/12/03 23:02:41.026 EST <Geode Membership View Creator> tid=0x347] total
weight lost in this view change is 65 of 111.  Quorum has been lost!
```

Log Level: warn

Category: Membership

Meaning: This message requires **URGENT** attention. It is closely associated with other messages, but indicates that the membership is very unhealthy, and you have potentially lost your entire cluster, or are having some "split brain" behavior, etc.

The above example message shows that a total weight of 65 has been lost, out of 111. This is greater than 50% of the weight, in one view change, hence driving the loss of quorum. When this much weight has been lost, it is generally something affecting the network connectivity, versus a GC event. Please read our extensive documentation on member weight, network partitions, etc.

Potential Resolutions:

It depends mostly on how many members have been removed, and it is possible that the entire cluster has gone down as a result of this loss of quorum. If you have `enable-network-partition-detection=true`, as we recommend, it is possible to lose the entire cluster if you see the above message. If most of the membership weight has crashed, for example, the losing side doesn't know that, but the losing side (i.e. the side with less weight) will shut itself down, even though it includes the only still running members. Restart members to restore your cluster to full health, and determine the root cause for why so many members crashed simultaneously.

possible loss of quorum due to the loss of <n> cache processes

Log Message:

```
[fatal 2021/12/03 23:02:41.027 EST <Geode Membership View Creator> tid=0x347]
Possible loss of quorum due to the loss of 6 cache processes: [<list of the ip's and
processes>]
```

Log Level: fatal

Category: Membership

Meaning: This is very closely tied to the "quorum has been lost" message. They will often go hand in hand, and potentially even out of order, where you will see the "possible loss" after the "has been lost" message.

Potential Resolutions:

Follow the guidance provided in the, "quorum has been lost," message. We definitely recommend having enable-network-partition-detection=true set to protect you from split brain driving the data in your split (now 2) distributed systems from diverging and becoming unrecoverable without manual intervention.

Membership service failure: Exiting due to possible network partition event due to loss of cache processes

Log Message:

```
[fatal 2021/12/03 23:02:42.028 EST <Geode Membership View Creator> tid=0x347]
Membership service failure: Exiting due to possible network partition event due to
loss of 6 cache processes: [<list of the 6 cache processes lost, in this example>]
```

Note: This message generally comes with a full stack trace showing the forceDisconnect.

Log Level: fatal

Category: Membership

Meaning: This message requires **URGENT** attention. It is closely associated with other loss of quorum messages, but indicates that the membership is very unhealthy, and you have potentially lost your entire cluster, or are having some “split brain” behavior, etc.

Potential Resolutions:

Follow the guidance provided in the, “quorum has been lost,” message. We definitely recommend having enable-network-partition-detection=true set to protect you from split brain driving the data in your split (now 2) distributed systems diverging and becoming unrecoverable without manual intervention. Do some research to determine whether some network event drove the VMware GemFire cluster into this state due to an inability to communicate across the distributed system.

<member> had a weight of <n>

Log Message:

```
[info 2021/12/09 23:19:55.100 EST memberXXX <Geode Membership View Creator> tid=0x57]
memberXXX)<v36>:50001 had a weight of 10
```

Log Level: info

Category: Membership

Meaning: This message indicates that a member has either crashed, or has been kicked out of the distributed system. By default, locators have a weight of 3, LEAD cache server has a weight of 15, and other cache servers have a weight of 10. In the example message, given the weight of 10, you would know that the member that has been kicked out is a non-lead cache server. Depending on your topology, and the number of members in your distributed system, the loss of one such cache server may not impact you much at all.

Potential Resolutions:

You certainly want to understand the cause of the member leaving the distributed system. If you have auto reconnect enabled, as you would by default, the member may rejoin automatically, unless it is a crash. If the member was kicked out due to being unresponsive, it may have auto-reconnected, restoring you to a fully running cluster. That said, you likely need to run a rebalance to evenly distribute data, or primary buckets if using partitioned regions. You may require GC tuning, etc. If you do not understand the cause of the membership change, you should reach out for assistance.

An additional Function Execution Processor thread is being launched

Log Message:

```
[warn 2021/12/01 21:29:56.689 EST memberXXX2 <Function Execution Processor1>
tid=0x27] An additional Function Execution Processor thread is being launched because
all <n> thread pool threads are in use for greater than <t> ms
```

Log Level: warn

Category: Operations

Meaning: This requires some action to achieve optimal behavior. If you see this message, it means that your normal behavior requires more than the configured number of function execution threads, set using DistributionManager.MAX_FE_THREADS. The default has increased recently, but if you see this message, regardless of the current setting shown in the example message, it indicates that your function executions will potentially take longer, due to VMware GemFire behavior.

Potential Resolutions:

If you see this message, then you should increase your DistributionManager.MAX_FE_THREADS configured setting, ,until you have eliminated such messages. You may want to consider the same for your DistributionManager.MAX_THREADS and DistributionManager.MAX_PR_THREADS settings, if not recently updated based on your current operations and load in the system.

Sending new view

Received new view

Admitting member

Log Message:

```
[info 2021/11/20 00:05:31.381 EST gbe-louweblps175(8551)-locator1 <Geode Membership
View Creator> tid=0xd34] sending new view View[<coordinator member
info>:48414:locator)<ec><v314>:41000|342] members: [<current member list>] shutdown:
[<member that shut down>:42306)<v315>:41003]
[info 2021/09/29 01:41:30.472 EDT <memberXXX> tid=0x1d] received new view:
View[<coordinator member>)<ec><v0>:50000|5] members: [list of members, indicating
shutdown, crashed members] old view is: <previous view information, including list of
members and state>

[info 2021/12/01 21:36:21.966 EST DCS-DCS-CLUSTER-10.104.39.130-dmnode-002 <View
Message Processor1> tid=0x3c] Admitting member <<memberXXX:26842)<v6>:10131>. Now
there are 6 non-admin member(s).
```

Log Level: info

Category: Membership

Meaning: These messages can be very helpful to understand who the coordinator of the Distributed System is, the lead cache server member, and the change in state of the membership, whether members are leaving or joining the distributed system. This will include the cause of leaving, whether a “graceful” shutdown, or a “crash”. You will only ever see the “Sending new view” message in the current coordinator of the system at that time. All members receive this view, and admit the member to the membership list. You only have a full membership when you see the expected number of non-admin members, six in the above, “Now there are 6 non-admin member(s),” example.

Potential Resolutions:

These are informational only, but if you do see unexpected membership changes, which drive these “new view” messages, you can search the logs for these messages to see whether it was considered graceful, a crash, etc., and look for other logging messages which likely provide additional insight.

Member at <memberIP> unexpectedly left the distributed cache

Log Message:

```
[info 2022/01/11 04:35:34.242 EST <View Message Processor1> tid=0x89] Member at
<memberXXX>:3853)<v11>:10104 unexpectedly left the distributed cache: departed
membership view
```

Log Level: info

Category: Membership

Meaning: This message is an indication that a member has experienced a non-graceful removal from the distributed system. This will then correspond with “new view” messages being sent to all members of the DS, showing the member in the list of “crashed” members.

Potential Resolutions:

This specific message doesn’t tell you much other than the change in ownership. Search for other messages across the cluster which may indicate the reason, such as being unresponsive. Perhaps it’s due to not responding to “heartbeat” messages. WIth auto reconnect, it is possible that the membership has been restored to a full membership, but it’s also important to check on the balance of data and load. A rebalance may be prudent to restore the balance in the system. This includes redistributing primary buckets for partitioned regions, which is generally a good idea after any changes in ownership, when time permits.

Cache server: failed accepting client connection

Remote host closed connection during handshake

SSL peer shut down incorrectly

Log Message:

```
[warn 2021/12/01 21:26:11.216 memberXXX <Handshaker /10.104.39.130:10102 Thread 8831>
tid=0x2d19e] Cache server: failed accepting client connection javax.net.ssl.SSLHandsha
keException: Remote host closed connection during handshake
javax.net.ssl.SSLHandshakeException: Remote host closed connection during handshake
        at sun.security.ssl.SSLSocketImpl.readRecord(SSLSocketImpl.java:994)
        at sun.security.ssl.SSLSocketImpl.performInitialHandshake(SSLSocketImpl.java:1
367)
        at sun.security.ssl.SSLSocketImpl.startHandshake(SSLSocketImpl.java:1395)
        at sun.security.ssl.SSLSocketImpl.startHandshake(SSLSocketImpl.java:1379)
        at org.apache.geode.internal.net.SocketCreator.handshakeIfSocketIsSSL(SocketCr
eator.java:1094)
        at org.apache.geode.internal.cache.tier.sockets.AcceptorImpl.getCommunicationM
odeForNonSelector(AcceptorImpl.java:1559)
        at org.apache.geode.internal.cache.tier.sockets.AcceptorImpl.handleNewClientCo
nnection(AcceptorImpl.java:1431)
        at org.apache.geode.internal.cache.tier.sockets.AcceptorImpl.lambda$handOffNew
ClientConnection$4(AcceptorImpl.java:1342)
        at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1
149)
        at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:
624)
        at java.lang.Thread.run(Thread.java:748)
Caused by: java.io.EOFException: SSL peer shut down incorrectly
        at sun.security.ssl.InputRecord.read(InputRecord.java:505)
        at sun.security.ssl.SSLSocketImpl.readRecord(SSLSocketImpl.java:975)
```

Log Level: warn

Category: Membership

Meaning: While this looks to be very SSL/TLS specific, this message is often driven by the many of the same client connectivity issues as in the non-SSL/TLS case. This is a client-server connection that is failing because the connection terminated. Besides the general client-server connectivity issues, however, this could also be caused when the client can’t validate the server’s Certificate, and so hangs up. This message does not indicate any reasons for why that connectivity was lost, but does indicate client-server connectivity issues and the cause needs to be investigated and understood.

Potential Resolutions:

Review client logs to see if there’s anything informative there, such as SSL/TLS validation issues, and then investigate logs and stats for possible connectivity or performance issues on the server.

Function: <functionName> cannot be executed because the members [list of members] are running low on memory

Log Message:

```
[error 2022/01/11 03:49:41.307 EST <ServerConnection on port 10230 Thread 5>
tid=0x28d] <function info> Function: <functionName> cannot be executed because the
members [<list of members>)<v3>:10104] are running low on memory
```

Log Level: error

Category: Operations, Storage

Meaning: This is very similar to the “canceled” query message, but applies to function executions. Essentially, before execution the system recognizes the heap has surpassed the critical-threshold in some subset of members, and therefore the system chooses not to begin the function execution. You should also see the “above heap critical threshold” message in some logs if seeing this message.

Potential Resolutions:

Please follow the same guidelines as the “Query execution canceled due to memory threshold” message.

Region <regionName> bucket <n> has persistent data that is no longer online stored at these locations

Log Message:

```
[error 2022/01/11 03:51:41.809 EST <ServerConnection on port 10230 Thread 2>
tid=0x21f] <filtered>:Region <regionName> bucket 51 has persistent data that is no
longer online stored at these locations: [<list of members hosting the bucket
including timestamp information>l]
```

Log Level: error

Category: Membership

Meaning: This message tells us that we have lost access to some persistent copy of the given bucket (“51” in the above example). So we know we have a partitioned persistent region where some of the hosting members are not available.

Potential Resolutions:

Determine the cause of the loss of the given member or members hosting that bucket, provided in the message. We do not recommend executing any gfsh “revoke” command without expert interaction and assistance. It is possible you could cause a loss of data.

Region has potentially stale data. Buckets [list] are waiting for another offline member

Log Message:

```
[info 2021/12/03 06:52:17.226 EST <PersistentBucketRecoverer for region <r>>
tid=0x147] Region <r> (and any colocated sub-regions) has potentially stale data.
Buckets [27, 85, 92] are waiting for another offline member to recover the latest
data.My persistent id is:
  DiskStore ID: <disk store id>
  Name: <member name>
  Location: <member location>
Offline members with potentially new data:[
  DiskStore ID: <disk store id of member with potentially newer data>
  Location: <member location>
  Buckets: [27, 85, 92]
] Use the gfsh show missing-disk-stores command to see all disk stores that are being
waited on by other members.
```

Log Level: info

Category: Membership

Meaning: This message normally shows when a member is starting, during bucket recovery of partitioned regions, and indicates that it is waiting for other members, where the data is considered to be more current, to start. Once those members start, the latest copy of the data will be accessible and the member will perform a GII (get initial image) to recover the buckets allowing the member to proceed with initialization.

Potential Resolutions:

If this message appears, perhaps you have members not yet started, and you need to start those members. We recommend starting all cache server processes simultaneously, especially after a clean shutdown, so that the subsequent startup has access to all buckets, and no member is stuck waiting for other members to start.

If you did not have a clean shutdown, or some member has been down for a long time, do NOT start up that member as the first member of a cluster. Otherwise, you will get into a ConflictingDatePersistenceException state that will then require revoking disk stores.

This is a completely avoidable scenario. It is better to start all of the members that have been up and part of the healthy cluster first, and then add back that member later, to be able to get that member up to date, with the latest copies of the buckets loaded from other members. If you see this message, you may want to check current status with the gfsh "show metrics" command to determine whether your number of buckets without redundancy is changing for the specified region over time. If not, you should definitely take a thread dump across all members to determine whether you are having some form of distributed deadlock issue during startup. It is possible that you are simply having major contention/congestion due to some insufficient configuration, such as DistributionManager.MAX_PR_THREAD or DistributionManager.MAX_THREADS. This can be evaluated by analyzing the statistics of the system using a tool like VSD.

Developing with VMware GemFire

Developing with VMware GemFire explains main concepts of application programming with VMware GemFire. It describes how to plan and implement regions, data serialization, event handling, delta propagation, transactions, and more.

For information about VMware GemFire REST application development, see Developing REST Applications for VMware GemFire.

- **Region Data Storage and Distribution**

  The VMware GemFire data storage and distribution models put your data in the right place at the right time. You should understand all the options for data storage in VMware GemFire before you start configuring your data regions.

- **Partitioned Regions**

  In addition to basic region management, partitioned regions include options for high availability, data location control, and data balancing across the cluster.

- **Distributed and Replicated Regions**

  In addition to basic region management, distributed and replicated regions include options for things like push and pull distribution models, global locking, and region entry versions to ensure consistency across VMware GemFire members.

- **Consistency for Region Updates**

  VMware GemFire ensures that all copies of a region eventually reach a consistent state on all members and clients that host the region, including VMware GemFire members that distribute region events.

- **General Region Data Management**

  For all regions, you have options to control memory use, back up your data to disk, and keep stale data out of your cache.

- **Data Serialization**

  Data that you manage in VMware GemFire must be serialized and deserialized for storage and transmittal between processes. You can choose among several options for data serialization.

- **Events and Event Handling**

  VMware GemFire provides versatile and reliable event distribution and handling for your cached data and system member events.

- **Delta Propagation**

Delta propagation allows you to reduce the amount of data you send over the network by including only changes to objects rather than the entire object.

- **Querying**

  VMware GemFire provides a SQL-like querying language called OQL that allows you to access data stored in VMware GemFire regions.

- **Continuous Querying**

  Continuous querying continuously returns events that match the queries you set up.

- **Transactions**

  VMware GemFire provides a transactions API, with `begin`, `commit`, and `rollback` methods. These methods are much the same as the familiar relational database transactions methods.

- **Function Execution**

  A function is a body of code that resides on a server and that an application can invoke from a client or from another server without the need to send the function code itself. The caller can direct a data-dependent function to operate on a particular dataset, or can direct a data-independent function to operate on a particular server, member, or member group.

Region Data Storage and Distribution

The VMware GemFire data storage and distribution models put your data in the right place at the right time. You should understand all the options for data storage in VMware GemFire before you configure your data regions.

- **Storage and Distribution Options**

  VMware GemFire provides several models for data storage and distribution, including partitioned or replicated regions as well as distributed or non-distributed regions (local cache storage).

- **Region Types**

  Region types define region behavior within a single cluster. You have various options for region data storage and distribution.

- **Region Data Stores and Data Accessors**

  Understand the difference between members that store data for a region and members that act only as data accessors to the region.

- **Creating Regions Dynamically**

  You can dynamically create regions in your application code and automatically instantiate them on members of a cluster.

Storage and Distribution Options

VMware GemFire provides several models for data storage and distribution, including partitioned or replicated regions as well as distributed or non-distributed regions (local cache storage).

Peer-to-Peer Region Storage and Distribution

At its most general, data management means having current data available when and where your applications need it. In a properly configured VMware GemFire installation, you store your data in your local members and VMware GemFire automatically distributes it to the other members that need it according to your cache configuration settings. You may be storing very large data objects that require special consideration, or you may have a high volume of data requiring careful configuration to safeguard your application's performance or memory use. You may need to be able to explicitly lock some data during particular operations. Most data management features are available as configuration options, which you can specify either using the `gfsh` cluster configuration service, `cache.xml` file or the API. Once configured, VMware GemFire manages the data automatically. For example, this is how you manage data distribution, disk storage, data expiration activities, and data partitioning. A few features are managed at run-time through the API.

At the architectural level, data distribution runs between peers in a single cluster and between clients and servers.

- Peer-to-peer provides the core distribution and storage models, which are specified as attributes on the data regions.
- For client/server, you choose which data regions to share between the client and server tiers. Then, within each region, you can fine-tune the data that the server automatically sends to the client by subscribing to subsets.

Data storage in any type of installation is based on the peer-to-peer configuration for each individual cluster. Data and event distribution is based on a combination of the peer-to-peer and system-to-system configurations.

Storage and distribution models are configured through cache and region attributes. The main choices are partitioned, replicated, or just distributed. All server regions must be partitioned or replicated. Each region's `data-policy` and `subscription-attributes`, and its `scope` if it is not a partitioned region, interact for finer control of data distribution.

Storing Data in the Local Cache

To store data in your local cache, use a region `refid` with a `RegionShortcut` or `ClientRegionShortcut` that has local state. These automatically set the region `data-policy` to a non-empty policy. Regions without storage can send and receive event distributions without storing anything in your application heap. With the other settings, all entry operations received are stored locally.

Region Types

Region types define region behavior within a single cluster. You have various options for region data storage and distribution.

Within a VMware GemFire cluster, you can define distributed regions and non-distributed regions, and you can define regions whose data is spread across the cluster, and regions whose data is entirely contained in a single member.

Your choice of region type is governed in part by the type of application you are running. In particular, you need to use specific region types for your servers and clients for effective communication between the two tiers:

- Server regions are created inside a `Cache` by servers and are accessed by clients that connect to the servers from outside the server's cluster. Server regions must have region type partitioned or replicated. Server region configuration uses the `RegionShortcut` enum settings.
- Client regions are created inside a `ClientCache` by clients and are configured to distribute data and events between the client and the server tier. Client regions must have region type `local`. Client region configuration uses the `ClientRegionShortcut` enum settings.
- Peer regions are created inside a `Cache`. Peer regions may be server regions, or they may be regions that are not accessed by clients. Peer regions can have any region type. Peer region configuration uses the `RegionShortcut` enum settings.

When you configure a server or peer region using `gfsh` or with the `cache.xml` file, you can use *region shortcuts* to define the basic configuration of your region. A region shortcut provides a set of default configuration attributes that are designed for various types of caching architectures. You can then add additional configuration attributes as needed to customize your application. For more information and a complete reference of these region shortcuts, see Region Shortcuts Reference.

These are the primary configuration choices for each data region.

Region Type	Description	Best suited for...
Partitioned	System-wide setting for the data set. Data is divided into buckets across the members that define the region. For high availability, configure redundant copies so each bucket is stored in multiple members with one member holding the primary.	Server regions and peer regions • Very large data sets • High availability • Write performance • Partitioned event listeners and data loaders
Replicated (distributed)	Holds all data from the distributed region. The data from the distributed region is copied into the member replica region. Can be mixed with non-replication, with some members holding replicas and some holding non-replicas.	Server regions and peer regions • Read heavy, small datasets • Asynchronous distribution • Query performance
Distributed non-replicated	Data is spread across the members that define the region. Each member holds only the data it has expressed interest in. Can be mixed with replication, with some members holding replicas and some holding non-replicas.	Peer regions, but not server regions and not client regions • Asynchronous distribution • Query performance
Non-distributed (local)	The region is visible only to the defining member.	Client regions and peer regions • Data that is not shared between applications

Partitioned Regions

Partitioning is a good choice for very large server regions. Partitioned regions are ideal for data sets in the hundreds of gigabytes and beyond.

Note: Partitioned regions generally require more JDBC connections than other region types because each member that hosts data must have a connection.

Partitioned regions group your data into buckets, each of which is stored on a subset of all of the system members. Data location in the buckets does not affect the logical view - all members see the same logical data set.

Use partitioning for:

- **Large data sets**. Store data sets that are too large to fit into a single member, and all members will see the same logical data set. Partitioned regions divide the data into units of storage called buckets that are split across the members hosting the partitioned region data, so no member needs to host all of the region's data. VMware GemFire provides dynamic redundancy recovery and rebalancing of partitioned regions, making them the choice for large-scale data containers. More members in the system can accommodate more uniform balancing of the data across all host members, allowing system throughput (both gets and puts) to scale as new members are added.
- **High availability**. Partitioned regions allow you configure the number of redundant copies of your data that VMware GemFire should make. If a member fails, your data will be available without interruption from the remaining members that host a redundant copy of the data. No data loss occurs as long as the number of server failures does not exceed the number of redundant copies. Partitioned regions can also be persisted to disk for additional high availability.
- **Scalability**. Partitioned regions can scale to large amounts of data because the data is divided between the members available to host the region. Increase your data capacity dynamically by simply adding new members. Partitioned regions also allow you to scale your processing capacity. Because your entries are spread out across the members hosting the region, reads and writes to those entries are also spread out across those members.
- **Good write performance**. You can configure the number of copies of your data. The amount of data transmitted per write does not increase with the number of members. By contrast, with replicated regions, each write must be sent to every member that has the region replicated, so the amount of data transmitted per write increases with the number of members.

In partitioned regions, you can colocate keys within buckets and across multiple partitioned regions. You can also control which members store which data buckets.

Replicated Regions

Replicated regions provide the highest performance in terms of throughput and latency. Replication is a good choice for small to medium size server regions.

Use replicated regions for:

- **Small amounts of data required by all members of the cluster**. For example, currency rate information and mortgage rates.
- **Data sets that can be contained entirely in a single member**. Each replicated region holds the complete data set for the region

- **High performance data access**. Replication guarantees local access from the heap for application threads, providing the lowest possible latency for data access.
- **Asynchronous distribution**. All distributed regions, replicated and non-replicated, provide the fastest distribution speeds.

Distributed, Non-Replicated Regions

Distributed regions provide the same performance as replicated regions, but each member stores only data in which it has expressed an interest, either by subscribing to events from other members or by defining the data entries in its cache.

Use distributed, non-replicated regions for:

- **Peer regions, but not server regions or client regions**. Server regions must be either replicated or partitioned. Client regions must be local.
- **Data sets where individual members need only notification and updates for changes to a subset of the data**. In non-replicated regions, each member receives only update events for the data entries it has defined in the local cache.
- **Asynchronous distribution**. All distributed regions, replicated and non-replicated, provide the fastest distribution speeds.

Local Regions

Note: When created using the `ClientRegionShortcut` settings, client regions are automatically defined as local, since all client distribution activities go to and come from the server tier.

The local region has no peer-to-peer distribution activity.

Use local regions for:

- **Client regions**. Distribution is only between the client and server tier.
- **Private data sets for the defining member**. The local region is not visible to peer members.

Region Data Stores and Data Accessors

Understand the difference between members that store data for a region and members that act only as data accessors to the region.

In most cases, when you define a data region in a member’s cache, you also specify whether the member is a data store. Members that store data for a region are referred to as data stores or data hosts. Members that do not store data are referred to as accessor members, or empty members. Any member, store or accessor, that defines a region can access it, put data into it, and receive events from other members. To configure a region so the member is a data accessor, you use configurations that specify no local data storage for the region. Otherwise, the member is a data store for the region.

For server regions, suppress local data storage at region creation by specifying a region shortcut that contains the term “PROXY” in its name, such as `PARTITION_PROXY` or `REPLICATE_PROXY`.

For client regions, suppress local data storage at region creation by specifying the `PROXY` region shortcut. Do not use the `CACHING_PROXY` shortcut for this purpose, as it allows local data storage.

Creating Regions Dynamically

You can dynamically create regions in your application code and automatically instantiate them on members of a cluster.

Due to the number of options involved, most developers use functions to create regions dynamically in their applications, as described in this topic. Dynamic regions can also be created from the `gfsh` command line.

For a complete discussion of using VMware GemFire functions, see Function Execution. Functions use the `org.apache.geode.cache.execute.FunctionService` class.

If your application does not require partitioned regions, you can use the `org.apache.geode.cache.DynamicRegionFactory` class to dynamically create regions, or you can create them using the `<dynamic-region-factory>` element in the cache.xml file that defines the region. (You can create partitioned regions dynamically, but you cannot use the `DynamicRegionFactory` class or the `<dynamic-region-factory>` element to do it.)

Note: Use of the `DynamicRegionFactory` class (and the `<dynamic-region-factory>` element) are deprecated in favor of the `FunctionService` approach described here.

In the following example, the `CreateRegionFunction` class defines a function invoked on a server by a client using the `onServer()` method of the `FunctionService` class. This function call initiates region creation by putting an entry into the region attributes metadata region. The entry key is the region name and the value is the set of region attributes used to create the region.

```
#CreateRegionFunction.java

import org.apache.geode.cache.Cache;
import org.apache.geode.cache.DataPolicy;
import org.apache.geode.cache.Declarable;
import org.apache.geode.cache.Region;
import org.apache.geode.cache.RegionFactory;
import org.apache.geode.cache.Scope;

import org.apache.geode.cache.configuration.RegionConfig;

import org.apache.geode.cache.execute.Function;
import org.apache.geode.cache.execute.FunctionContext;

import java.util.Properties;

public class CreateRegionFunction implements Function<RegionConfig>, Declarable {

  private Region<String,RegionConfig> regionAttributesMetadataRegion;

  private static final String REGION_ATTRIBUTES_METADATA_REGION =
                                                     "_regionAttributesMetadata";

  public enum Status {SUCCESSFUL, UNSUCCESSFUL, ALREADY_EXISTS}

  public void execute(FunctionContext<RegionConfig> context) {
    RegionConfig regionConfig = context.getArguments();
    // Create or retrieve region
    Status status = createOrRetrieveRegion(context.getCache(), regionConfig);

    // Return status
    context.getResultSender().lastResult(status);
  }

  private Status createOrRetrieveRegion(Cache cache, RegionConfig regionConfig) {
    Status status = Status.SUCCESSFUL;
    String regionName = regionConfig.getName();
    Region<Object, Object> region = cache.getRegion(regionName);
    if (region == null) {
      // Put the attributes into the metadata region. The afterCreate call
      // creates the region.
      this.regionAttributesMetadataRegion.put(regionName, regionConfig);

      // Retrieve the region after creating it
      region = cache.getRegion(regionName);
      if (region == null) {
        status = Status.UNSUCCESSFUL;
      }
    } else {
      status = Status.ALREADY_EXISTS;
    }
    return status;
  }

  private void initializeRegionAttributesMetadataRegion(Cache cache) {
    this.regionAttributesMetadataRegion =
                                 cache.getRegion(REGION_ATTRIBUTES_METADATA_REGION);
    if (this.regionAttributesMetadataRegion == null) {
      RegionFactory<String, RegionConfig> factory = cache.createRegionFactory();
      factory.setDataPolicy(DataPolicy.REPLICATE);
      factory.setScope(Scope.DISTRIBUTED_ACK);
      factory.addCacheListener(new CreateRegionCacheListener());
      this.regionAttributesMetadataRegion =
                                  factory.create(REGION_ATTRIBUTES_METADATA_REGION);
    }
  }

  public String getId() {
    return getClass().getSimpleName();
  }

  public void initialize(Cache cache, Properties properties) {
    initializeRegionAttributesMetadataRegion(cache);
  }
}
```

The `CreateRegionCacheListener` class is a cache listener that implements two methods, `afterCreate()` and `afterRegionCreate()`. The `afterCreate()` method creates the region. The `afterRegionCreate()` method causes each new server to create all the regions defined in the metadata region.

```
#CreateRegionCacheListener.java

import org.apache.geode.cache.Cache;
import org.apache.geode.cache.Declarable;
import org.apache.geode.cache.EntryEvent;
import org.apache.geode.cache.Region;
import org.apache.geode.cache.RegionEvent;
import org.apache.geode.cache.RegionExistsException;

import org.apache.geode.cache.configuration.RegionConfig;

import org.apache.geode.cache.util.CacheListenerAdapter;

import java.util.Map;

public class CreateRegionCacheListener extends CacheListenerAdapter<String,RegionConfi
g> implements Declarable {

  public void afterCreate(EntryEvent<String,RegionConfig> event) {
    createRegion(event.getRegion().getCache(), event.getKey(), event.getNewValue());
  }

  public void afterRegionCreate(RegionEvent<String,RegionConfig> event) {
    Cache cache = event.getRegion().getCache();
    Region<String,RegionConfig> region = event.getRegion();
    for (Map.Entry<String,RegionConfig> entry : region.entrySet()) {
      createRegion(cache, entry.getKey(), entry.getValue());
    }
  }

  private void createRegion(Cache cache, String regionName, RegionConfig regionConfig)
{
    if (cache.getLogger().fineEnabled()) {
      cache.getLogger().fine("CreateRegionCacheListener creating region named=" + regi
onName + "; config: " + regionConfig);
    }
    Region<Object, Object> region = cache.getRegion(regionConfig.getName());
    if (region == null) {
      try {
        region = cache.createRegionFactory(regionConfig.getType()).create(regionConfi
g.getName());
        cache.getLogger().info("CreateRegionCacheListener created region=" + region);
      } catch (RegionExistsException e) {
        cache.getLogger().info("CreateRegionCacheListener region already exists region
=" + region);
      }
    } else {
      cache.getLogger().info("CreateRegionCacheListener region already exists region="
+ region);
    }
  }
}
```

Partitioned Regions

In addition to basic region management, partitioned regions include options for high availability, data location control, and data balancing across the cluster.

- **Understanding Partitioning**

To use partitioned regions, you should understand how they work and your options for managing them.

- **Configuring Partitioned Regions**

  Plan the configuration and ongoing management of your partitioned region for host and accessor members and configure the regions for startup.

- **Configuring the Number of Buckets for a Partitioned Region**

  Decide how many buckets to assign to your partitioned region and set the configuration accordingly.

- **Custom-Partitioning and Colocating Data**

  You can customize how VMware GemFire groups your partitioned region data with custom partitioning and data colocation.

- **Configuring High Availability for Partitioned Regions**

  By default, VMware GemFire stores only a single copy of your partitioned region data among the region's data stores. You can configure VMware GemFire to maintain redundant copies of your partitioned region data for high availability.

- **Configuring Single-Hop Client Access to Server-Partitioned Regions**

  Single-hop data access enables the client pool to track where a partitioned region's data is hosted in the servers. To access a single entry, the client directly contacts the server that hosts the key, in a single hop.

- **Rebalancing Partitioned Region Data**

  In a cluster with minimal contention to the concurrent threads reading or updating from the members, you can use rebalancing to dynamically increase or decrease your data and processing capacity.

- **Checking Redundancy in Partitioned Regions**

  Under some circumstances, it can be important to verify that your partitioned region data is redundant and that upon member restart, redundancy has been recovered properly across partitioned region members.

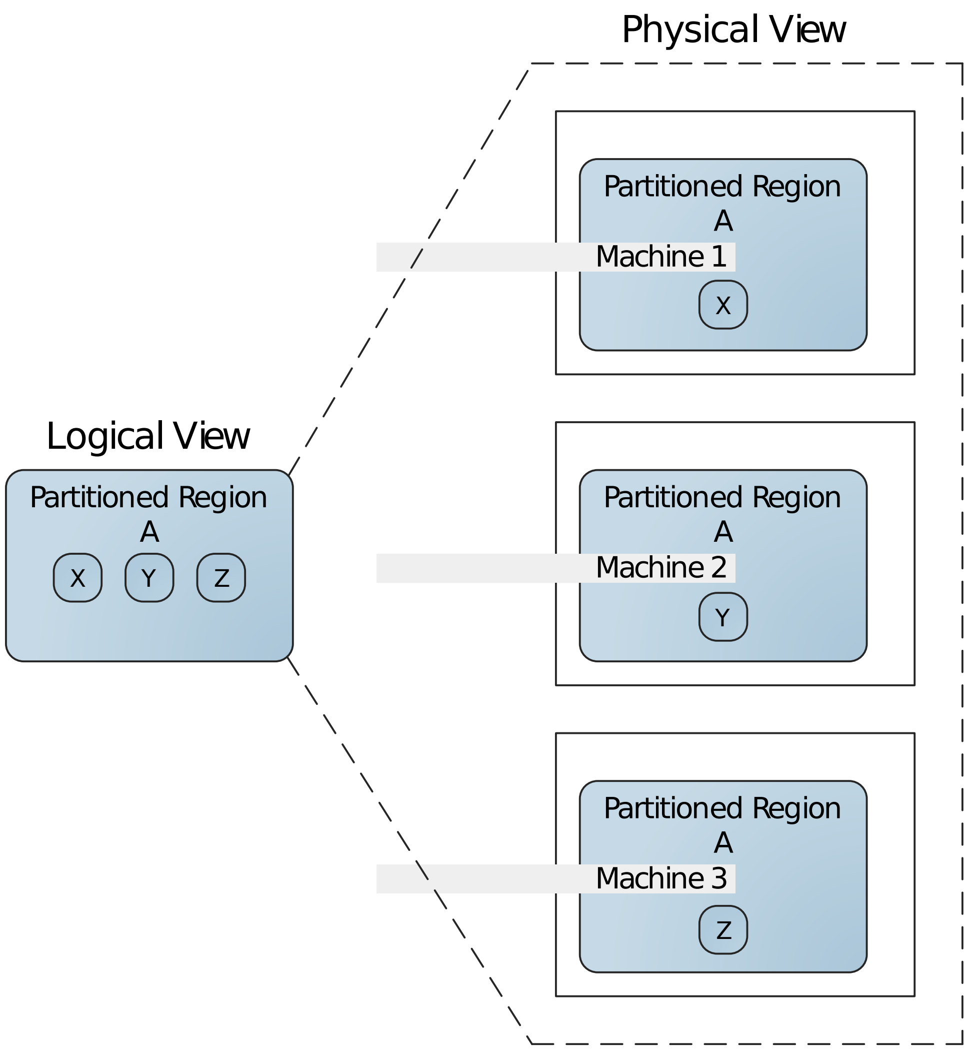
- **Moving Partitioned Region Data to Another Member**

  You can use the `PartitionRegionHelper` `moveBucketByKey` and `moveData` methods to explicitly move partitioned region data from one member to another.

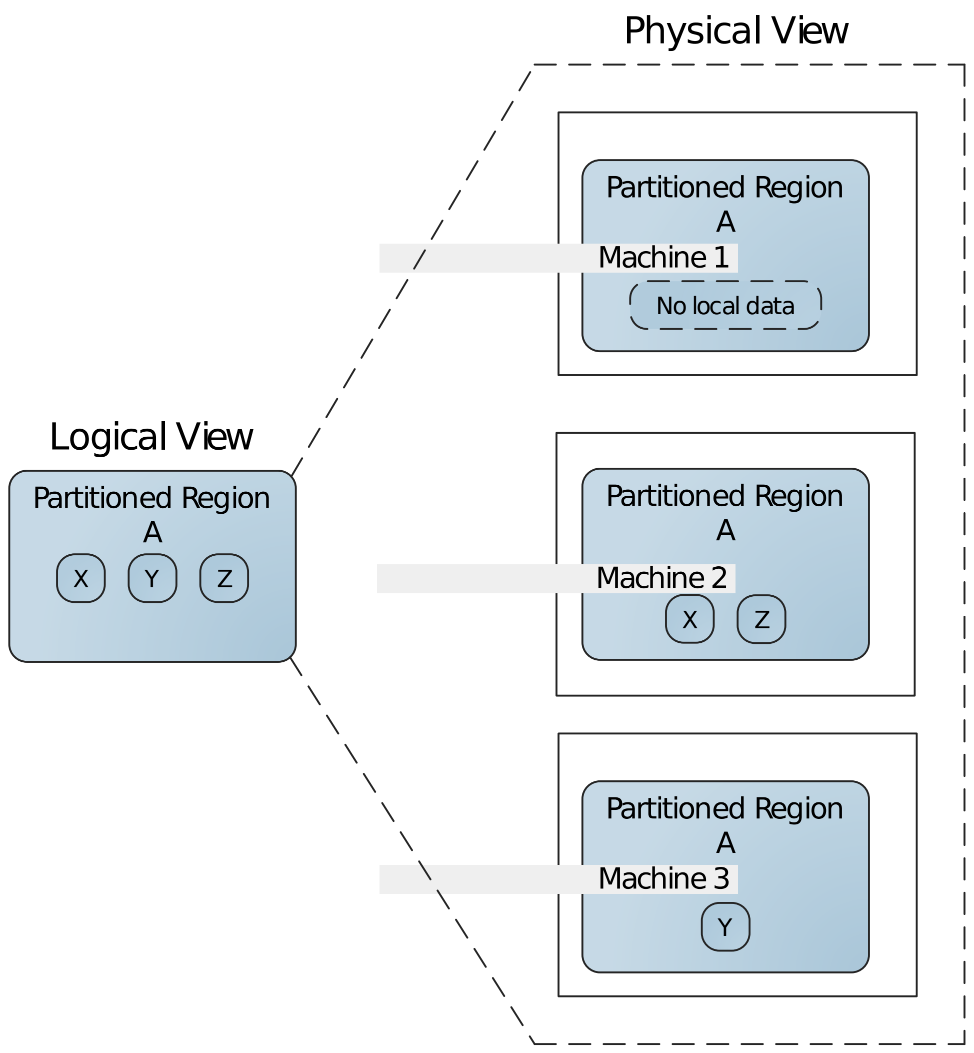
Understanding Partitioning

To use partitioned regions, you should understand how they work and your options for managing them.

During operation, a partitioned region looks like one large virtual region, with the same logical view held by all of the members where the region is defined.



For each member where you define the region, you can choose how much space to allow for region data storage, including no local storage at all. The member can access all region data regardless of how much is stored locally.



A cluster can have multiple partitioned regions, and it can mix partitioned regions with distributed regions and local regions. The usual requirement for unique region names, except for regions with local scope, still applies. A single member can host multiple partitioned regions.

Data Partitioning

VMware GemFire automatically determines the physical location of data in the members that host a partitioned region's data. VMware GemFire breaks partitioned region data into units of storage known as buckets and stores each bucket in a region host member. Buckets are distributed in accordance to the member's region attribute settings.

When an entry is created, it is assigned to a bucket. Keys are grouped together in a bucket and always remain there. If the configuration allows, the buckets may be moved between members to balance the load.

You must run the data stores needed to accommodate storage for the partitioned region's buckets. You can start new data stores on the fly. When a new data store creates the region, it takes responsibility for as many buckets as allowed by the partitioned region and member configuration.

You can customize how VMware GemFire groups your partitioned region data with custom partitioning and data colocation.

Partitioned Region Operation

A partitioned region operates much like a non-partitioned region with distributed scope. Most of the standard `Region` methods are available, although some methods that are normally local operations become distributed operations, because they work on the partitioned region as a whole instead of the local cache. For example, a `put` or `create` into a partitioned region may not actually be stored into the cache of the member that called the operation. The retrieval of any entry requires no more than one hop between members.

Partitioned regions support the client/server model, just like other regions. If you need to connect dozens of clients to a single partitioned region, using servers greatly improves performance.

Additional Information About Partitioned Regions

Keep the following in mind about partitioned regions:

- Partitioned regions never run asynchronously. Operations in partitioned regions always wait for acknowledgement from the caches containing the original data entry and any redundant copies.
- A partitioned region needs a cache loader in every region data store (`local-max-memory` > 0).
- VMware GemFire distributes the data buckets as evenly as possible across all members storing the partitioned region data, within the limits of any custom partitioning or data colocation that you use. The number of buckets allotted for the partitioned region determines the granularity of data storage and thus how evenly the data can be distributed. The number of buckets is a total for the entire region across the cluster.
- In rebalancing data for the region, VMware GemFire moves buckets, but does not move data around inside the buckets.
- You can query partitioned regions, but there are certain limitations. See Querying Partitioned Regions for more information.

Configuring Partitioned Regions

Plan the configuration and ongoing management of your partitioned region for host and accessor members and configure the regions for startup.

Before you begin, understand Basic Configuration and Programming.

1. Start your region configuration using one of the `PARTITION` region shortcut settings. See Region Shortcuts and Custom Named Region Attributes.
2. If you need high availability for your partitioned region, configure for that. See Configure High Availability for a Partitioned Region.
3. Estimate the amount of space needed for the region. If you use redundancy, this is the max for all primary and secondary copies stored in the member. For example, with redundancy

of one, each region data entry requires twice the space than with no redundancy, because the entry is stored twice. See Memory Requirements for Cached Data.

4. Configure the total number of buckets for the region. This number must be the same for colocated regions. See Configuring the Number of Buckets for a Partitioned Region.
5. Configure your members' data storage and data loading for the region:
    1. You can have members with no local data storage and members with varying amounts of storage. Determine the max memory available in your different member types for this region. These will be set in the `partition-attributes local-max-memory`. This is the only setting in `partition-attributes` that can vary between members. Use these max values and your estimates for region memory requirements to help you figure how many members to start out with for the region.
    2. For members that store data for the region (`local-max-memory` greater than 0), define a data loader. See Implement a Data Loader.
    3. If you have members with no local data storage (`local-max-memory` set to 0), review your system startup/shutdown procedures. Make sure there is always at least one member with local data storage running when any members with no storage are running.
6. If you want to custom partition the data in your region or colocate data between multiple regions, code and configure accordingly. See Understanding Custom Partitioning and Data Colocation.
7. Plan your partition rebalancing strategy and configure and program for that. See Rebalancing Partitioned Region Data.

Note: To configure a partitioned region using gfsh, see gfsh Command Help.

Configuring the Number of Buckets for a Partitioned Region

Decide how many buckets to assign to your partitioned region and set the configuration accordingly.

The total number of buckets for the partitioned region determines the granularity of data storage and thus how evenly the data can be distributed. VMware GemFire distributes the buckets as evenly as possible across the data stores. The number of buckets is fixed after region creation.

The partition attribute `total-num-buckets` sets the number for the entire partitioned region across all participating members. Set it using one of the following:

- XML:

```
<region name="PR1">
  <region-attributes refid="PARTITION">
    <partition-attributes total-num-buckets="7"/>
  </region-attributes>
</region>
```

- Java:

```
RegionFactory rf =
    cache.createRegionFactory(RegionShortcut.PARTITION);
rf.setPartitionAttributes(new PartitionAttributesFactory().setTotalNumBuckets
(7).create());
custRegion = rf.create("customer");
```

- gfsh:

  Use the --total-num-buckets parameter of the `create region` command. For example:

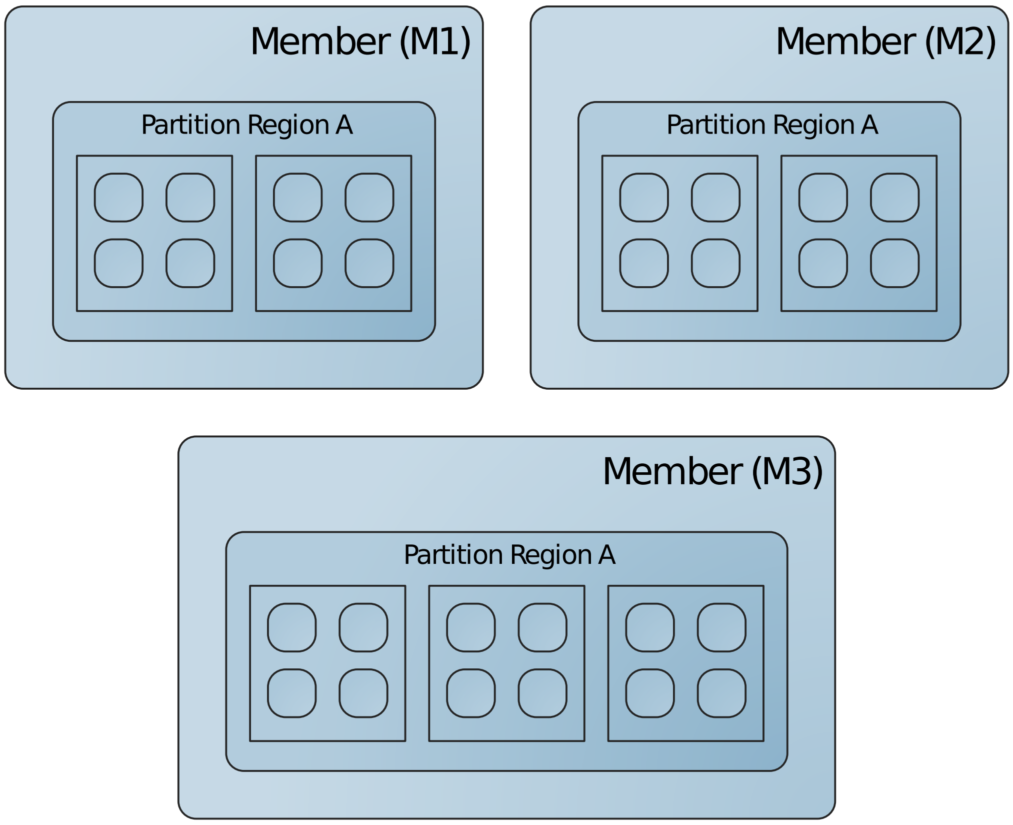
  ```
  gfsh>create region --name="PR1" --type=PARTITION --total-num-buckets=7
  ```

Calculate the Total Number of Buckets for a Partitioned Region

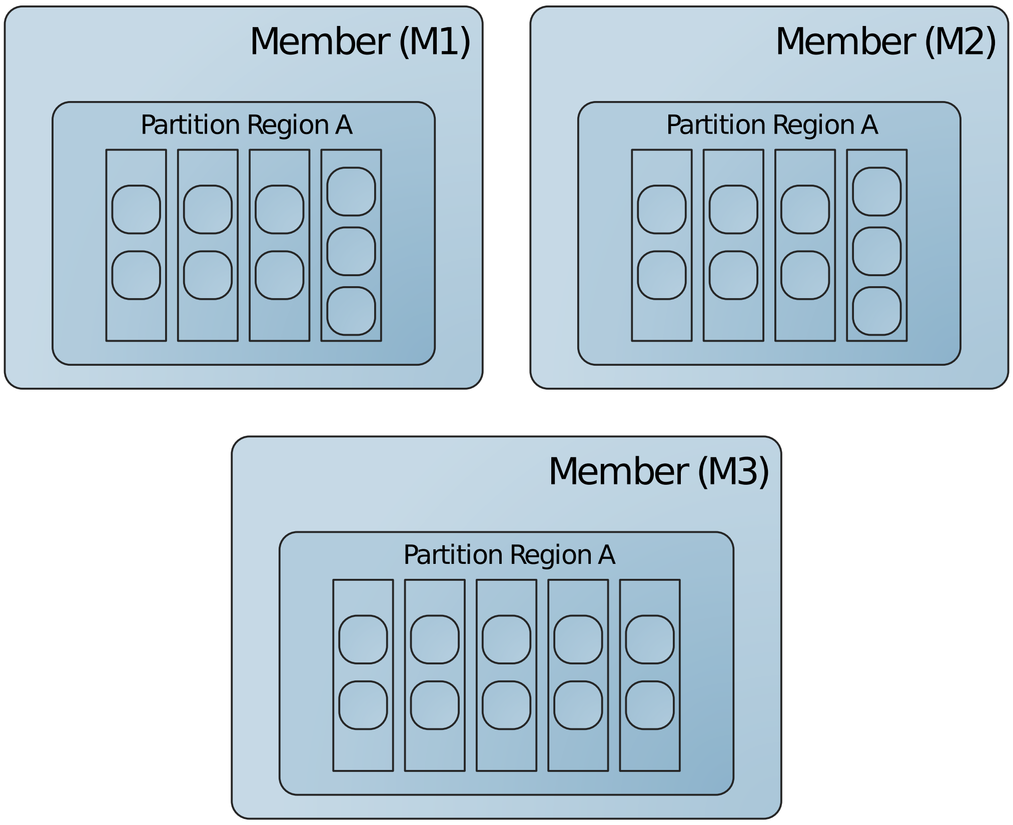
Follow these guidelines to calculate the total number of buckets for the partitioned region:

- Use a prime number. This provides the most even distribution.
- Make it at least four times as large as the number of data stores you expect to have for the region. The larger the ratio of buckets to data stores, the more evenly the load can be spread across the members. Note that there is a trade-off between load balancing and overhead, however. Managing a bucket introduces significant overhead, especially with higher levels of redundancy.

You are trying to avoid the situation where some members have significantly more data entries than others. For example, compare the next two figures. This figure shows a region with three data stores and seven buckets. If all the entries are accessed at about the same rate, this configuration creates a hot spot in member M3, which has about fifty percent more data than the other data stores. M3 is likely to be a slow receiver and potential point of failure.



Configuring more buckets gives you fewer entries in a bucket and a more balanced data distribution. This figure uses the same data as before but increases the number of buckets to 13. Now the data entries are distributed more evenly.



Custom-Partitioning and Colocating Data

You can customize how VMware GemFire groups your partitioned region data with custom partitioning and data colocation.

- **Understanding Custom Partitioning and Data Colocation**

  Custom partitioning and data colocation can be used separately or in conjunction with one another.

- **Standard Custom Partitioning**

  By default, VMware GemFire partitions each data entry into a bucket using a hashing policy on the key. Additionally, the physical location of the key-value pair is abstracted away from the application. You can change these policies for a partitioned region by providing a standard partition resolver that maps entries to a set of buckets called a partition.

- **Fixed Custom Partitioning**

  By default, VMware GemFire partitions each data entry into a bucket using a hashing policy on the key. Additionally, the physical location of the key-value pair is abstracted away from the application. You can change these policies for a partitioned region by providing a fixed partition resolver that not only maps entries to a set of buckets called a partition, but also specifies which members host which data buckets.

- **Colocate Data from Different Partitioned Regions**

  By default, VMware GemFire allocates the data locations for a partitioned region independent of the data locations for any other partitioned region. You can change this policy for any group of partitioned regions, so that cross-region, related data is all hosted by the same member. Colocation is required for some operations, and it increases performance for others by reducing the number of data accesses to entries that are hosted on other cluster members.

Understanding Custom Partitioning and Data Colocation

Custom partitioning and data colocation can be used separately or in conjunction with one another.

Custom Partitioning

Use custom partitioning to group like entries into region buckets within a region. By default, VMware GemFire assigns new entries to buckets based on the entry key’s hash code. With custom partitioning, you can assign your entries to buckets in whatever way you want.

You can generally get better performance if you use custom partitioning to group similar data within a region. For example, a query run on all accounts created in January runs faster if all January account data is hosted by a single member. Grouping all data for a single customer can improve performance of data operations that work on customer data. Data aware function execution also takes advantage of custom partitioning.

With custom partitioning, you have two choices:

- **Standard custom partitioning**. With standard custom partitioning, you group entries into buckets, but you do not specify where the buckets reside. VMware GemFire always keeps the entries in the buckets you have specified, but may move the buckets around for load balancing. See Standard Custom Partitioning for implementation and configuration details.
- **Fixed custom partitioning**. With fixed custom partitioning, you specify the exact member where each region entry resides. You assign an entry to a partition and then to a bucket within that partition. You name specific members as primary and secondary hosts of each partition.

  This gives you complete control over the locations of your primary and any secondary buckets for the region. This can be useful when you want to store specific data on specific physical machines or when you need to keep data close to certain hardware elements.

  Fixed partitioning has these requirements and caveats:

  - VMware GemFire cannot rebalance fixed partition region data, because it cannot move the buckets around among the host members. You must carefully consider your expected data loads for the partitions you create.
  - With fixed partitioning, the region configuration is different between host members. Each member identifies the named partitions it hosts, and whether it is hosting the primary copy or a secondary copy. You then program a fixed-partition resolver to return the partition id, so the entry is placed on the right members. Only one member can be primary for a particular partition name, and that member cannot be the partition’s secondary.

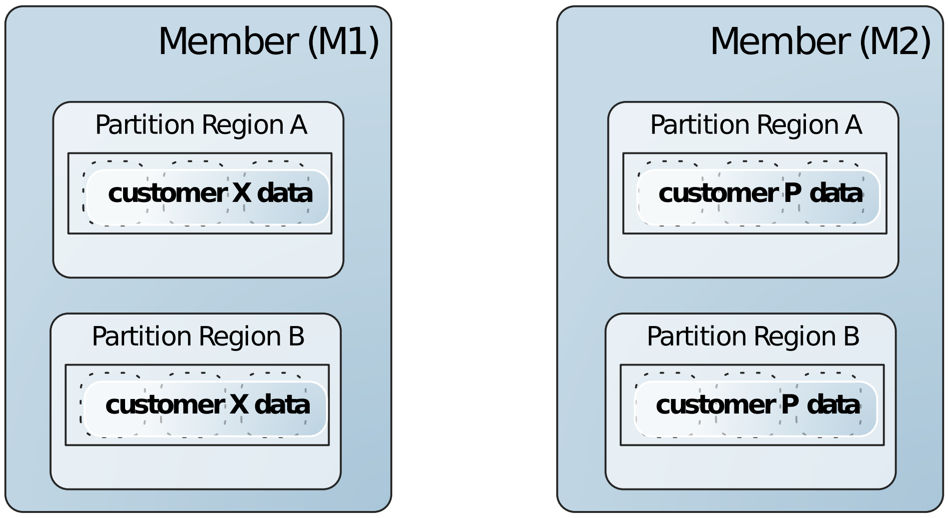
  See Fixed Custom Partitioning for implementation and configuration details.

Data Colocation Between Regions

With data colocation, VMware GemFire stores entries that are related across multiple data regions in a single member. VMware GemFire does this by storing all of the regions' buckets with the same ID together in the same member. During rebalancing operations, VMware GemFire moves these bucket groups together or not at all.

So, for example, if you have one region with customer contact information and another region with customer orders, you can use colocation to keep all contact information and all orders for a single customer in a single member. This way, any operation done for a single customer uses the cache of only a single member.

This figure shows two regions with data colocation where the data is partitioned by customer type.



Data colocation requires the same data partitioning mechanism for all of the colocated regions. You can use the default partitioning provided by VMware GemFire or any of the custom partitioning strategies.

You must use the same high availability settings across your colocated regions.

See Colocate Data from Different Partitioned Regions for implementation and configuration details.

Standard Custom Partitioning

By default, VMware GemFire partitions each data entry into a bucket using a hashing policy on the key. Additionally, the physical location of the key-value pair is abstracted away from the application. You can change these policies for a partitioned region by providing a standard custom partition resolver that maps entries in a custom manner.

Note: If you are both colocating region data and custom partitioning, all colocated regions must use the same custom partitioning mechanism. See Colocate Data from Different Partitioned Regions.

To custom-partition your region data, follow two steps:

- implement the interface
- configure the regions

Implementing Standard Custom Partitioning

- Implement the `org.apache.geode.cache.PartitionResolver` interface in one of the following ways, listed here in the search order used by VMware GemFire:
    - **Using a custom class**. Implement the `PartitionResolver` within your class, and then specify your class as the partition resolver during region creation.
    - **Using the key's class**. Have the entry key's class implement the `PartitionResolver` interface.
    - **Using the callback argument's class**. Have the class that implements your callback argument implement the `PartitionResolver` interface. When using this implementation, any and all `Region` operations must be those that specify the callback argument.
- Implement the resolver's `getName`, `init`, and `close` methods.

  A simple implementation of `getName` is

  ```
  return getClass().getName();
  ```

  The `init` method does any initialization steps upon cache start that relate to the partition resolver's task. This method is only invoked when using a custom class that is configured by `gfsh` or by XML (in a `cache.xml` file).

  The `close` method accomplishes any clean up that must be accomplished before a cache close completes. For example, `close` might close files or connections that the partition resolver opened.
- Implement the resolver's `getRoutingObject` method to return the routing object for each entry. A hash of that returned routing object determines the bucket. Therefore, `getRoutingObject` should return an object that, when run through its `hashCode`, directs grouped objects to the desired bucket.

  Note: Only the key, as returned by `getKey`, should be used by `getRoutingObject`. Do not use the value associated with the key or any additional metadata in the implementation of `getRoutingObject`. Do not use `getOperation` or `getValue`.

Implementing the String Prefix Partition Resolver

VMware GemFire provides an implementation of a string-based partition resolver in `org.apache.geode.cache.util.StringPrefixPartitionResolver`. This resolver does not require any further implementation. It groups entries into buckets based on a portion of the key. All keys must be strings, and each key must include a '|' character, which delimits the string. The substring that precedes the '|' delimiter within the key will be returned by `getRoutingObject`.

Configuring the Partition Resolver Region Attribute

Configure the region so VMware GemFire finds your resolver for all region operations.

- **Custom class**. Use one of these methods to specify the partition resolver region attribute:

  gfsh:

  Add the option `--partition-resolver` to the `gfsh create region` command, specifying the package and class name of the standard custom partition resolver.

  gfsh for the String Prefix Partition Resolver:

Add the option `--partition-resolver=org.apache.geode.cache.util.StringPrefixPartitionResolver` to the `gfsh create region` command.

Java API:

```
PartitionResolver resolver = new TradesPartitionResolver();
PartitionAttributes attrs =
    new PartitionAttributesFactory()
    .setPartitionResolver(resolver).create();

Cache c = new CacheFactory().create();

Region r = c.createRegionFactory()
    .setPartitionAttributes(attrs)
    .create("trades");
```

Java API for the String Prefix Partition Resolver:

```
PartitionAttributes attrs =
    new PartitionAttributesFactory()
    .setPartitionResolver(new StringPrefixPartitionResolver()).create();

Cache c = new CacheFactory().create();

Region r = c.createRegionFactory()
    .setPartitionAttributes(attrs)
    .create("customers");
```

XML:

```
<region name="trades">
    <region-attributes>
        <partition-attributes>
            <partition-resolver>
                <class-name>myPackage.TradesPartitionResolver
                </class-name>
            </partition-resolver>
        <partition-attributes>
    </region-attributes>
</region>
```

XML for the String Prefix Partition Resolver:

```
<region name="customers">
    <region-attributes>
        <partition-attributes>
            <partition-resolver>
                <class-name>org.apache.geode.cache.util.StringPrefixPartitionRe
solver
                </class-name>
            </partition-resolver>
        <partition-attributes>
    </region-attributes>
</region>
```

- If your colocated data is in a server system, add the `PartitionResolver` implementation class to the `CLASSPATH` of your Java clients. For Java single-hop access to work, the resolver class needs to have a zero-argument constructor, and the resolver class must not have any state; the `init` method is included in this restriction.

Fixed Custom Partitioning

By default, VMware GemFire partitions each data entry into a bucket using a hashing policy on the key. Additionally, the physical location of the key-value pair is abstracted away from the application. You can change these policies for a partitioned region by providing a fixed custom partition resolver that not only maps entries to a set of buckets called a partition, but also specifies which members host which data buckets.

Note: If you are both colocating region data and custom partitioning, all colocated regions must use the same custom partitioning mechanism. See Colocate Data from Different Partitioned Regions.

To custom-partition your region data, follow two steps:

- implement the interface
- configure the regions

These steps differ based on which partition resolver is used.

Implementing Fixed Custom Partitioning

- Implement the `org.apache.geode.cache.FixedPartitionResolver` interface within one of the following locations, listed here in the search order used by VMware GemFire:
  - **Custom class**. Specify this class as the partition resolver during region creation.
  - **Entry key**. For keys implemented as objects, define the interface for the key's class.
  - **Within the cache callback class**. Implement the interface within a cache callback's class. When using this implementation, any and all `Region` operations must be those that specify the callback as a parameter.
- Implement the resolver's `getName`, `init`, and `close` methods.

  A simple implementation of `getName` is

  ```
  return getClass().getName();
  ```

  The `init` method does any initialization steps upon cache start that relate to the partition resolver's task.

  The `close` method accomplishes any clean up that must be accomplished before a cache close completes. For example, `close` might close files or connections that the partition resolver opened.
- Implement the resolver's `getRoutingObject` method to return the routing object for each entry. A hash of that returned routing object determines the bucket within a partition.

  This method can be empty for fixed partitioning where there is only one bucket per partition. That implementation assigns partitions to servers such that the application has full control of grouping entries on servers.

Note: Only fields on the key should be used when creating the routing object. Do not use the value or additional metadata for this purpose.

- Implement the `getPartitionName` method to return the name of the partition for each entry, based on where you want the entries to reside. All entries within a partition will be on a single server.

  This example places the data based on date, with a different partition name for each quarter-year and a different routing object for each month.

```
/**
 * Returns one of four different partition names
 * (Q1, Q2, Q3, Q4) depending on the entry's date
 */
class QuarterFixedPartitionResolver implements
    FixedPartitionResolver<String, String> {

  @Override
  public String getPartitionName(EntryOperation<String, String> opDetails,
      Set<String> targetPartitions) {

     Date date = (Date)opDetails.getKey();
     Calendar cal = Calendar.getInstance();
     cal.setTime(date);
     int month = cal.get(Calendar.MONTH);
     if (month >= 0 && month < 3) {
        if (targetPartitions.contains("Q1")) return "Q1";
     }
     else if (month >= 3 && month < 6) {
        if (targetPartitions.contains("Q2")) return "Q2";
     }
     else if (month >= 6 && month < 9) {
        if (targetPartitions.contains("Q3")) return "Q3";
     }
     else if (month >= 9 && month < 12) {
        if (targetPartitions.contains("Q4")) return "Q4";
     }
     return "Invalid Quarter";
  }

  @Override
  public String getName() {
     return "QuarterFixedPartitionResolver";
  }

  @Override
  public Serializable getRoutingObject(EntryOperation<String, String> opDetail
s) {
     Date date = (Date)opDetails.getKey();
     Calendar cal = Calendar.getInstance();
     cal.setTime(date);
     int month = cal.get(Calendar.MONTH);
     return month;
  }

  @Override
  public void close() {
    }
}
```

Configuring Fixed Custom Partitioning

- Set the fixed-partition attributes for each member.

  These attributes define the data stored for the region by the member and must be different for different members. See `org.apache.geode.cache.FixedPartitionAttributes` for definitions of the attributes. Define each `partition-name` in your data-host members for the region. For each partition name, in the member you want to host the primary copy, define it with `is-primary` set to `true`. In every member you want to host the secondary copy, define it with `is-primary` set to `false` (the default). The number of secondaries must match the number of redundant copies you have defined for the region. See Configure High Availability for a Partitioned Region.

  Note: Buckets for a partition are hosted only by the members that have defined the partition name in their `FixedPartitionAttributes`.

  These examples set the partition attributes for a member to be the primary host for the “Q1” partition data and a secondary host for “Q3” partition data. - XML:

  ~~~
  ``` pre
  <cache>
     <region name="Trades">
        <region-attributes>
           <partition-attributes redundant-copies="1">
             <partition-resolver>
                <class-name>myPackage.QuarterFixedPartitionResolver</class-name>
             </partition-resolver>
             <fixed-partition-attributes partition-name="Q1" is-primary="true"/>
             <fixed-partition-attributes partition-name="Q3" is-primary="false"
                  num-buckets="6"/>
           </partition-attributes>
        </region-attributes>
     </region>
  </cache>
  ```
  ~~~

  - Java:

    ```
    FixedPartitionAttribute fpa1 = FixedPartitionAttributes
         .createFixedPartition("Q1", true);
    FixedPartitionAttribute fpa3 = FixedPartitionAttributes
         .createFixedPartition("Q3", false, 6);

    PartitionAttributesFactory paf = new PartitionAttributesFactory()
         .setPartitionResolver(new QuarterFixedPartitionResolver())
         .setTotalNumBuckets(12)
         .setRedundantCopies(2)
         .addFixedPartitionAttribute(fpa1)
         .addFixedPartitionAttribute(fpa3);

    Cache c = new CacheFactory().create();

    Region r = c.createRegionFactory()
    ```

```
    .setPartitionAttributes(paf.create())
    .create("Trades");
```

- gfsh:

  You cannot specify a fixed partition resolver using gfsh.

- If your colocated data is in a server system, add the class that implements the `FixedPartitionResolver` interface to the `CLASSPATH` of your Java clients. For Java single-hop access to work, the resolver class needs to have a zero-argument constructor, and the resolver class must not have any state; the `init` method is included in this restriction.

Colocate Data from Different Partitioned Regions

By default, VMware GemFire allocates the data locations for a partitioned region independent of the data locations for any other partitioned region. You can change this policy for any group of partitioned regions, so that cross-region, related data is all hosted by the same member. Colocation is required for some operations, and it increases performance for others by reducing the number of data accesses to entries that are hosted on other cluster members.

Data colocation between partitioned regions generally improves the performance of data-intensive operations. You can reduce network hops for iterative operations on related data sets. Compute-heavy applications that are data-intensive can significantly increase overall throughput. For example, a query run on a patient's health records, insurance, and billing information is more efficient if all data is grouped in a single member. Similarly, a financial risk analytical application runs faster if all trades, risk sensitivities, and reference data associated with a single instrument are together.

Procedure

1. Identify one region as the central region, with which data in the other regions is explicitly colocated. If you use persistence for any of the regions, you must persist the central region.
   1. Create the central region before you create the others, either in the `cache.xml` or your code. Regions in the XML are created before regions in the code, so if you create any of your colocated regions in the XML, you must create the central region in the XML before the others. VMware GemFire will verify its existence when the others are created and return `IllegalStateException` if the central region is not there. Do not add any colocation specifications to this central region.
   2. For all other regions, in the region partition attributes, provide the central region's name in the `colocated-with` attribute. Use one of these methods:
      - XML:

```
<cache>
    <region name="trades">
        <region-attributes>
            <partition-attributes>
                ...
            <partition-attributes>
        </region-attributes>
    </region>
    <region name="trade_history">
        <region-attributes>
```

```
            <partition-attributes colocated-with="trades">
                ...
            <partition-attributes>
        </region-attributes>
    </region>
</cache>
```

- Java:

```
PartitionAttributes attrs = ...
Region trades = new RegionFactory().setPartitionAttributes(attrs)
    .create("trades");
...
attrs = new PartitionAttributesFactory().setColocatedWith(trades.g
etFullPath())
    .create();
Region trade_history = new RegionFactory().setPartitionAttributes
(attrs)
    .create("trade_history");
```

- gfsh:

```
gfsh>create region --name="trades" type=PARTITION
gfsh> create region --name="trade_history" --colocated-with="trade
s"
```

2. For each of the colocated regions, use the same values for these partition attributes related to bucket management:
    - `recovery-delay`
    - `redundant-copies`
    - `startup-recovery-delay`
    - `total-num-buckets`
3. If you custom partition your region data, specify the custom resolver for all colocated regions. This example uses the same partition resolver for both regions:
    - XML:

```
<cache>
    <region name="trades">
        <region-attributes>
            <partition-attributes>
            <partition-resolver name="TradesPartitionResolver">
                <class-name>myPackage.TradesPartitionResolver
                </class-name>
            <partition-attributes>
        </region-attributes>
    </region>
    <region name="trade_history">
        <region-attributes>
            <partition-attributes colocated-with="trades">
            <partition-resolver name="TradesPartitionResolver">
                <class-name>myPackage.TradesPartitionResolver
                </class-name>
            <partition-attributes>
        </region-attributes>
    </region>
</cache>
```

- Java:

  ```
  PartitionResolver resolver = new TradesPartitionResolver();
  PartitionAttributes attrs =
      new PartitionAttributesFactory()
      .setPartitionResolver(resolver).create();
  Region trades = new RegionFactory().setPartitionAttributes(attrs)
      .create("trades");
  attrs = new PartitionAttributesFactory()
      .setColocatedWith(trades.getFullPath()).setPartitionResolver(resolve
  r)
      .create();
  Region trade_history = new RegionFactory().setPartitionAttributes(attrs)
      .create("trade_history");
  ```

- gfsh:

  Specify a partition resolver as described in the configuration section of Custom-Partition Your Region Data.

4. If you want to persist data in the colocated regions, persist the central region and then persist the other regions as needed. Use the same disk store for all of the colocated regions that you persist.

Configuring High Availability for Partitioned Regions

By default, VMware GemFire stores only a single copy of your partitioned region data among the region's data stores. You can configure VMware GemFire to maintain redundant copies of your partitioned region data for high availability.

- **Understanding High Availability for Partitioned Regions**

  With high availability, each member that hosts data for the partitioned region gets some primary copies and some redundant (secondary) copies.

- **Configure High Availability for a Partitioned Region**

  Configure in-memory high availability for your partitioned region. Set other high-availability options, like redundancy zones and redundancy recovery strategies.

Understanding High Availability for Partitioned Regions

With high availability, each member that hosts data for the partitioned region gets some primary copies and some redundant (secondary) copies.

With redundancy, if one member fails, operations continue on the partitioned region with no interruption of service:

- If the member hosting the primary copy is lost, VMware GemFire makes a secondary copy the primary. This might cause a temporary loss of redundancy, but not a loss of data.

- Whenever there are not enough secondary copies to satisfy redundancy, the system works to recover redundancy by assigning another member as secondary and copying the data to it.

Note: You can still lose cached data when you are using redundancy if enough members go down in a short enough time span.

You can configure how the system works to recover redundancy when it is not satisfied. You can configure recovery to take place immediately or, if you want to give replacement members a chance to start up, you can configure a wait period. Redundancy recovery is also automatically attempted during any partitioned data rebalancing operation. Use the `gemfire.MAX_PARALLEL_BUCKET_RECOVERIES` system property to configure the maximum number of buckets that are recovered in parallel. By default, up to 8 buckets are recovered in parallel any time the system attempts to recover redundancy.

Without redundancy, the loss of any of the region’s data stores causes the loss of some of the region’s cached data. Generally, you should not use redundancy when your applications can directly read from another data source, or when write performance is more important than read performance.

Controlling Where Your Primaries and Secondaries Reside

By default, VMware GemFire places your primary and secondary data copies for you, avoiding placement of two copies on the same physical machine. If there are not enough machines to keep different copies separate, VMware GemFire places copies on the same physical machine. You can change this behavior, so VMware GemFire only places copies on separate machines.

You can also control which members store your primary and secondary data copies. VMware GemFire provides two options:

- **Fixed custom partitioning**. This option is set for the region. Fixed partitioning gives you absolute control over where your region data is hosted. With fixed partitioning, you provide VMware GemFire with the code that specifies the bucket and data store for each data entry in the region. When you use this option with redundancy, you specify the primary and secondary data stores. Fixed partitioning does not participate in rebalancing because all bucket locations are fixed by you.
- **Redundancy zones**. This option is set at the member level. Redundancy zones let you separate primary and secondary copies by member groups, or zones. You assign each data host to a zone. Then VMware GemFire places redundant copies in different redundancy zones, the same as it places redundant copies on different physical machines. You can use this to split data copies across different machine racks or networks, This option allows you to add members on the fly and use rebalancing to redistribute the data load, with redundant data maintained in separate zones. When you use redundancy zones, VMware GemFire will not place two copies of the data in the same zone, so make sure you have enough zones.

Running Processes in Virtual Machines

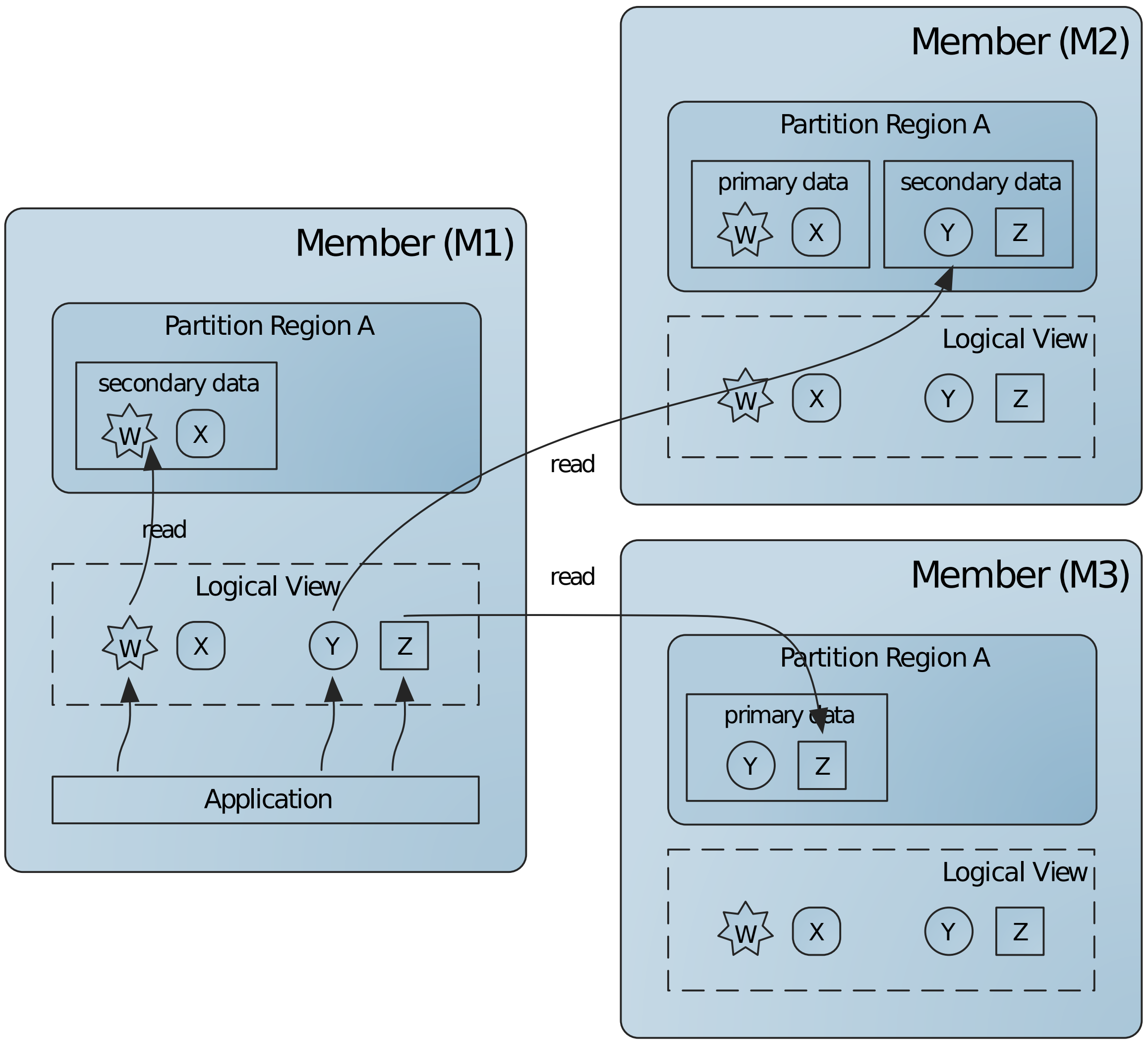
By default, VMware GemFire stores redundant copies on different machines. When you run your processes in virtual machines, the normal view of the machine becomes the VM and not the physical machine. If you run multiple VMs on the same physical machine, you could end up storing partitioned region primary buckets in separate VMs, but on the same physical machine as your secondaries. If the physical machine fails, you can lose data. When you run in VMs, you can configure VMware GemFire to identify the physical machine and store redundant copies on different physical machines.

Reads and Writes in Highly-Available Partitioned Regions

VMware GemFire treats reads and writes differently in highly-available partitioned regions than in other regions because the data is available in multiple members:

- Write operations (like `put` and `create`) go to the primary for the data keys and then are distributed synchronously to the redundant copies. Events are sent to the members configured with `subscription-attributes` `interest-policy` set to `all`.
- Read operations go to any member holding a copy of the data, with the local cache favored, so a read intensive system can scale much better and handle higher loads.

In this figure, M1 is reading W, Y, and Z. It gets W directly from its local copy. Since it doesn't have a local copy of Y or Z, it goes to a cache that does, picking the source cache at random.



Configure High Availability for a Partitioned Region

Configure in-memory high availability for your partitioned region. Set other high-availability options, like redundancy zones and redundancy recovery strategies.

Here are the main steps for configuring high availability for a partitioned region. See later sections for details.

1. Set the number of redundant copies the system should maintain of the region data. See Set the Number of Redundant Copies.
2. (Optional) If you want to group your data store members into redundancy zones, configure them accordingly. See Configure Redundancy Zones for Members.
3. (Optional) If you want VMware GemFire to only place redundant copies on different physical machines, configure for that. See Set Enforce Unique Host.
4. Decide how to manage redundancy recovery and change VMware GemFire's default behavior as needed.
    - **After a member crashes**. If you want automatic redundancy recovery, change the configuration for that. See Configure Member Crash Redundancy Recovery for a Partitioned Region.
    - **After a member joins**. If you do *not* want immediate, automatic redundancy recovery, change the configuration for that. See Configure Member Join Redundancy Recovery for a Partitioned Region.
5. Decide how many buckets VMware GemFire should attempt to recover in parallel when performing redundancy recovery. By default, the system recovers up to 8 buckets in parallel. Use the `gemfire.MAX_PARALLEL_BUCKET_RECOVERIES` system property to increase or decrease the maximum number of buckets to recover in parallel any time redundancy recovery is performed.
6. For all but fixed partitioned regions, review the points at which you kick off rebalancing. Redundancy recovery is done automatically at the start of any rebalancing. This is most important if you run with no automated recovery after member crashes or joins. See Rebalancing Partitioned Region Data.

During runtime, you can add capacity by adding new members for the region. For regions that do not use fixed partitioning, you can also kick off a rebalancing operation to spread the region buckets among all members.

- **Set the Number of Redundant Copies**

  Configure in-memory high availability for your partitioned region by specifying the number of secondary copies you want to maintain in the region's data stores.

- **Configure Redundancy Zones for Members**

  Group members into redundancy zones so VMware GemFire will separate redundant data copies into different zones.

- **Set Enforce Unique Host**

  Configure VMware GemFire to use only unique physical machines for redundant copies of partitioned region data.

- **Configure Member Crash Redundancy Recovery for a Partitioned Region**

  Configure whether and how redundancy is recovered in a partition region after a member crashes.

- **Configure Member Join Redundancy Recovery for a Partitioned Region**

  Configure whether and how redundancy is recovered in a partition region after a member joins.

Set the Number of Redundant Copies

Configure in-memory high availability for your partitioned region by specifying the number of secondary copies you want to maintain in the region's data stores.

Specify the number of redundant copies you want for your partitioned region data in the partition attribute `redundant-copies` setting. The default setting is 0.

For example:

- XML:

```
<region name="PR1">
  <region-attributes refid="PARTITION">
    <partition-attributes redundant-copies="1"/>
  </region-attributes>
</region>
```

- Java:

```
PartitionAttributes pa =
    new PartitionAttributesFactory().setRedundantCopies(1).create();
```

- gfsh:

```
gfsh>create region --name="PR1" --type=PARTITION --redundant-copies=1
```

Configure Redundancy Zones for Members

Group members into redundancy zones so VMware GemFire will separate redundant data copies into different zones.

Understand how to set a member's `gemfire.properties` settings. See Reference.

Group your partition region hosts into redundancy zones with the `gemfire.properties` setting `redundancy-zone`.

For example, if you had redundancy set to 1, so you have one primary and one secondary copy of each data entry, you could split primary and secondary data copies between two machine racks by defining one redundancy zone for each rack. To do this, you set this zone in the `gemfire.properties` for all members that run on one rack:

```
redundancy-zone=rack1
```

You would set this zone `gemfire.properties` for all members on the other rack:

```
redundancy-zone=rack2
```

Each secondary copy would be hosted on the rack opposite the rack where its primary copy is hosted.

Set Enforce Unique Host

Configure VMware GemFire to use only unique physical machines for redundant copies of partitioned region data.

Understand how to set a member's `gemfire.properties` settings. See Reference.

Configure your members so VMware GemFire always uses different physical machines for redundant copies of partitioned region data using the `gemfire.properties` setting `enforce-unique-host`. The default for this setting is false.

Example:

```
enforce-unique-host=true
```

Configure Member Crash Redundancy Recovery for a Partitioned Region

Configure whether and how redundancy is recovered in a partition region after a member crashes.

Use the partition attribute `recovery-delay` to specify member crash redundancy recovery.

recovery-delay partition attribute	Effect following a member failure
-1	No automatic recovery of redundancy following a member failure. This is the default.
long greater than or equal to 0	Number of milliseconds to wait after a member failure before recovering redundancy.

By default, redundancy is not recovered after a member crashes. If you expect to quickly restart most crashed members, combining this default setting with member join redundancy recovery can help you avoid unnecessary data shuffling while members are down. By waiting for lost members to rejoin, redundancy recovery is done using the newly started members and partitioning is better balanced with less processing.

Set crash redundancy recovery using one of the following:

- XML:

  ```
  // Give a crashed member 10 seconds to restart
  // before recovering redundancy
  <region name="PR1">
    <region-attributes refid="PARTITION">
      <partition-attributes recovery-delay="10000"/>
    </region-attributes>
  </region>
  ```

- Java:

```
PartitionAttributes pa = new PartitionAttributesFactory().setRecoveryDelay(1000
0).create();
```

- gfsh:

```
gfsh>create region --name="PR1" type=PARTITION --recovery-delay=10000
```

Configure Member Join Redundancy Recovery for a Partitioned Region

This section covers configuring whether and how redundancy is recovered in a partitioned region, after a member joins.

Use the partition attribute `startup-recovery-delay` to specify member join redundancy recovery.

value of `startup-recovery-delay`	Effect following a member join
-1	No automatic recovery of redundancy after a new member comes online. With this value and the default `recovery-delay` setting, redundancy recovery is only achieved by a rebalance operation.
long >= 0	Number of milliseconds to wait after a member joins before recovering redundancy. The default is 0 (zero), which causes immediate redundancy recovery whenever a member that hosts the partitioned region joins.

Setting `startup-recovery-delay` to a value higher than the default of 0 allows multiple new members to join before redundancy recovery begins. With the multiple members present during recovery, the system will spread redundancy recovery among them. With no delay, if multiple members are started in close succession, the system may choose only the first member started for most or all of the redundancy recovery.

Note: Satisfying redundancy is not the same as adding capacity. If redundancy is satisfied, new members do not take buckets until the invocation of a rebalance operation.

The parallel recovery implementation recovers quickly. For this reason, it is even more important to configure `startup-recovery-delay` to an appropriate value when restarting multiple members at the same time. Set `startup-recovery-delay` to a value that ensures all members are up and available *before* redundancy recovery kicks in.

Set join redundancy recovery using one of the following:

- XML:

```
// Wait 5 seconds after a new member joins before
// recovering redundancy
<region name="PR1">
  <region-attributes refid="PARTITION">
    <partition-attributes startup-recovery-delay="5000"/>
  </region-attributes>
</region>
```

- Java:

```
PartitionAttributes pa = new PartitionAttributesFactory().setStartupRecoveryDel
ay(5000).create();
```

- gfsh:

```
gfsh>create region --name="PR1" --type=PARTITION --startup-recovery-delay=5000
```

Configuring Single-Hop Client Access to Server-Partitioned Regions

Single-hop data access enables the client pool to track where a partitioned region's data is hosted in the servers. To access a single entry, the client directly contacts the server that hosts the key, in a single hop.

- **Understanding Client Single-Hop Access to Server-Partitioned Regions**

  With single-hop access the client connects to every server, so more connections are generally used. This works fine for smaller installations, but is a barrier to scaling.

- **Configure Client Single-Hop Access to Server-Partitioned Regions**

  Configure your client/server system for direct, single-hop access to partitioned region data in the servers.

Understanding Client Single-Hop Access to Server-Partitioned Regions

With single-hop access the client connects to every server, so more connections are generally used. This works fine for smaller installations, but is a barrier to scaling.

If you have a large installation with many clients, you may want to disable single hop by setting the pool attribute, `pr-single-hop-enabled` to false in your pool declarations.

Without single hop, the client uses whatever server connection is available, the same as with all other operations. The server that receives the request determines the data location and contacts the host, which might be a different server. So more multiple-hop requests are made to the server system.

Note: Single hop is used for the following operations: `put`, `get`, `destroy`, `invalidate`, `putAll`, `getAll`, `removeAll` and `onRegion` function execution.

Even with single hop access enabled, you will occasionally see some multiple-hop behavior. To perform single-hop data access, clients automatically get metadata from the servers about where the entry buckets are hosted. The metadata is maintained lazily. It is only updated after a single-hop operation ends up needing multiple hops, an indicator of stale metadata in the client.

Single Hop and the Pool max-connections Setting

Do not set the pool's `max-connections` setting with single hop enabled. Limiting the pool's connections with single hop can cause connection thrashing, throughput loss, and server log bloat.

If you need to limit the pool's connections, either disable single hop or keep a close watch on your system for these negative effects.

Setting no limit on connections, however, can result in too many connections to your servers, possibly causing you to run up against your system's file handle limits. Review your anticipated connection use and make sure your servers are able to accommodate it.

Balancing Single-Hop Server Connection Use

Single-hop gives the biggest benefits when data access is well balanced across your servers. In particular, the loads for client/server connections can get out of balance if you have these in combination:

- Servers that are empty data accessors or that do not host the data the clients access through single-key operations
- Many single-key operations from the clients

If data access is greatly out of balance, clients can thrash trying to get to the data servers. In this case, it might be faster to disable single hop and go through servers that do not host the data.

Configure Client Single-Hop Access to Server-Partitioned Regions

Configure your client/server system for direct, single-hop access to partitioned region data in the servers.

This requires a client/server installation that uses one or more partitioned regions on the server.

1. Verify the client's pool attribute, `pr-single-hop-enabled` is not set or is set to true. It is true by default. 2.

   If possible, leave the pool's `max-connections` at the default unlimited setting (-1).

2. If possible, use a custom data resolver to partition your server region data according to your clients' data use patterns. See Custom-Partition Your Region Data. Include the server's partition resolver implementation in the client's `CLASSPATH`. The server passes the name of the resolver for each custom partitioned region, so the client uses the proper one. If the server does not use a partition resolver, the default partitioning between server and client matches, so single hop works. 4.

   Add single-hop considerations to your overall server load balancing plan. Single-hop uses data location rather than least loaded server to pick the servers for single-key operations. Poorly balanced single-hop data access can affect overall client/server load balancing. Some counterbalancing is done automatically because the servers with more single-key operations become more loaded and are less likely to be picked for other operations.

Rebalancing Partitioned Region Data

In a cluster with minimal contention to the concurrent threads reading or updating from the members, you can use rebalancing to dynamically increase or decrease your data and processing capacity.

Rebalancing is a member operation. It affects all partitioned regions defined by the member, regardless of whether the member hosts data for the regions. The rebalancing operation performs two tasks:

1. If the configured partition region redundancy is not satisfied, rebalancing does what it can to recover redundancy. See Configure High Availability for a Partitioned Region.
2. Rebalancing moves the partitioned region data buckets between host members as needed to establish the most fair balance of data and behavior across the cluster.

For efficiency, when starting multiple members, trigger the rebalance a single time, after you have added all members.

Note: If you have transactions running in your system, be careful in planning your rebalancing operations. Rebalancing may move data between members, which could cause a running transaction to fail with a `TransactionDataRebalancedException`. Fixed custom partitioning prevents rebalancing altogether. All other data partitioning strategies allow rebalancing and can result in this exception unless you run your transactions and your rebalancing operations at different times.

Kick off a rebalance using one of the following:

- `gfsh` command. First, starting a `gfsh` prompt and connect to the cluster. Then type the following command:

  ```
  gfsh>rebalance
  ```

  Optionally, you can specify regions to include or exclude from rebalancing, specify a time-out for the rebalance operation or just simulate a rebalance operation. Type `help rebalance` or see rebalance for more information.

- API call:

  ```
  ResourceManager manager = cache.getResourceManager();
  RebalanceOperation op = manager.createRebalanceFactory().start();
  //Wait until the rebalance is complete and then get the results
  RebalanceResults results = op.getResults();
  //These are some of the details we can get about the run from the API
  System.out.println("Took " + results.getTotalTime() + " milliseconds\n");
  System.out.println("Transfered " + results.getTotalBucketTransferBytes()+ "byte
  s\n");
  ```

You can also just simulate a rebalance through the API, to see if it's worth it to run:

```
ResourceManager manager = cache.getResourceManager();
RebalanceOperation op = manager.createRebalanceFactory().simulate();
RebalanceResults results = op.getResults();
System.out.println("Rebalance would transfer " + results.getTotalBucketTransferBytes()
+" bytes ");
System.out.println(" and create " + results.getTotalBucketCreatesCompleted() + " bucke
ts.\n");
```

How Partitioned Region Rebalancing Works

The rebalancing operation runs asynchronously.

By default, rebalancing is performed on one partitioned region at a time. For regions that have colocated data, the rebalancing works on the regions as a group, maintaining the data colocation between the regions.

You can optionally rebalance multiple regions in parallel by setting the `gemfire.resource.manager.threads` system property. Setting this property to a value greater than 1 enables VMware GemFire to rebalance multiple regions in parallel, any time a rebalance operation is initiated using the API.

You can continue to use your partitioned regions normally while rebalancing is in progress. Read operations, write operations, and function executions continue while data is moving. If a function is executing on a local data set, you may see a performance degradation if that data moves to another host during function execution. Future function invocations are routed to the correct member.

VMware GemFire tries to ensure that each member has the same percentage of its available space used for each partitioned region. The percentage is configured in the `partition-attributes local-max-memory` setting.

Partitioned region rebalancing:

- Does not allow the `local-max-memory` setting to be exceeded unless LRU eviction is enabled with overflow to disk.
- Places multiple copies of the same bucket on different host IP addresses whenever possible.
- Resets entry time to live and idle time statistics during bucket migration.
- Replaces offline members.

When to Rebalance a Partitioned Region

You typically want to trigger rebalancing when capacity is increased or reduced through member startup, shut down or failure.

You may also need to rebalance when:

- You use redundancy for high availability and have configured your region to not automatically recover redundancy after a loss. In this case, VMware GemFire only restores redundancy when you invoke a rebalance. See Configure High Availability for a Partitioned Region.
- You have uneven hashing of data. Uneven hashing can occur if your keys do not have a hash code method, which ensures uniform distribution, or if you use a `PartitionResolver` to colocate your partitioned region data (see Colocate Data from Different Partitioned Regions). In either case, some buckets may receive more data than others. Rebalancing can be used to even out the load between data stores by putting fewer buckets on members that are hosting large buckets.

How to Simulate Region Rebalancing

You can simulate the rebalance operation before moving any actual data around by executing the `rebalance` command with the following option:

```
gfsh>rebalance --simulate
```

Note: If you are using `heap_lru` for data eviction, you may notice a difference between your simulated results and your actual rebalancing results. This discrepancy can be due to the VM starting to evict entries after you execute the simulation. Then when you perform an actual rebalance operation, the operation will make different decisions based on the newer heap size.

Automated Rebalancing

The experimental automated rebalance feature triggers a rebalance operation based on a time schedule.

Checking Redundancy in Partitioned Regions

Under some circumstances, it can be important to verify that your partitioned region data is redundant and that upon member restart, redundancy has been recovered properly across partitioned region members.

You can verify partitioned region redundancy by making sure that the `numBucketsWithoutRedundancy` statistic is **zero** for all your partitioned regions. To check this statistic, use the following `gfsh` command:

```
gfsh>show metrics --categories=partition --region=region_name
```

For example:

```
gfsh>show metrics --categories=partition --region=posts

Cluster-wide Region Metrics
--------- | --------------------------- | -----
partition | putLocalRate                | 0
          | putRemoteRate               | 0
          | putRemoteLatency            | 0
          | putRemoteAvgLatency         | 0
          | bucketCount                 | 1
          | primaryBucketCount          | 1
          | numBucketsWithoutRedundancy | 1
          | minBucketSize               | 1
          | maxBucketSize               | 0
          | totalBucketSize             | 1
          | averageBucketSize           | 1
```

If you have `start-recovery-delay=-1` configured for your partitioned region, you will need to perform a rebalance on your region after you restart any members in your cluster in order to recover redundancy.

If you have `start-recovery-delay` set to a low number, you may need to wait extra time until the region has recovered redundancy.

Moving Partitioned Region Data to Another Member

You can use the `PartitionRegionHelper` `moveBucketByKey` and `moveData` methods to explicitly move partitioned region data from one member to another.

The `moveBucketByKey` method moves the bucket that contains the specified key from a source member to a destination member. For example, you could use the method to move a popular product item to a new, empty member to reduce load on the source member.

For example:

```
Object product = ...
Region r = ...
DistributedSystem ds = ...
String memberName = ...

//Find the member that is currently hosting the product.
Set<DistributedMember> sourceMembers =
PartitionRegionHelper.getAllMembersForKey(r, product);

//Find the member to move the product to.
DistributedMember destination = ds.findDistributedMember(memberName);

//In this example we assume there is always at least one source.
//In practice, you should check that at least one source
//for the data is available.
source = sourceMembers.iterator().next();

//Move the bucket to the new node. The bucket will
//be moved when this method completes. It throws an exception
//if there is a problem or invalid arguments.
PartitionRegionHelper.moveBucketByKey(r, source, destination, product);
```

See the Java API documentation for `org.apache.geode.cache.partition.PartitionRegionHelper.moveBucketByKey` for more details.

The `moveData` method moves data up to a given percentage (measured in bytes) from a source member to a destination member. For example, you could use this method to move a specified percentage of data from an overloaded member to another member to improve distribution.

For example:

```
Region r = ...
DistributedSystem ds = ...
String sourceName = ...
String destName = ...

//Find the source member.
DistributedMember source = ds.findDistributedMember(sourceName);
DistributedMember destination = ds.findDistributedMember(destName);

//Move up to 20% of the data from the source to the destination node.
PartitionRegionHelper.moveData(r, source, destination, 20);
```

See the Java API documentation for `org.apache.geode.cache.partition.PartitionRegionHelper.moveData` for more details.

For more information on partitioned regions and rebalancing, see Partitioned Regions.

Distributed and Replicated Regions

In addition to basic region management, distributed and replicated regions include options for things like push and pull distribution models, global locking, and region entry versions to ensure consistency across VMware GemFire members.

- **How Distribution Works**

  To use distributed and replicated regions, you should understand how they work and your options for managing them.

- **Options for Region Distribution**

  You can use distribution with and without acknowledgment, or global locking for your region distribution. Regions that are configured for distribution with acknowledgment can also be configured to resolve concurrent updates consistently across all VMware GemFire members that host the region.

- **How Replication and Preloading Work**

  To work with replicated and preloaded regions, you should understand how their data is initialized and maintained in the cache.

- **Configure Distributed, Replicated, and Preloaded Regions**

  Plan the configuration and ongoing management of your distributed, replicated, and preloaded regions, and configure the regions.

- **Locking in Global Regions**

  In global regions, the system locks entries and the region during updates. You can also explicitly lock the region and its entries as needed by your application. Locking includes system settings that help you optimize performance and locking behavior between your members.

How Distribution Works

To use distributed and replicated regions, you should understand how they work and your options for managing them.

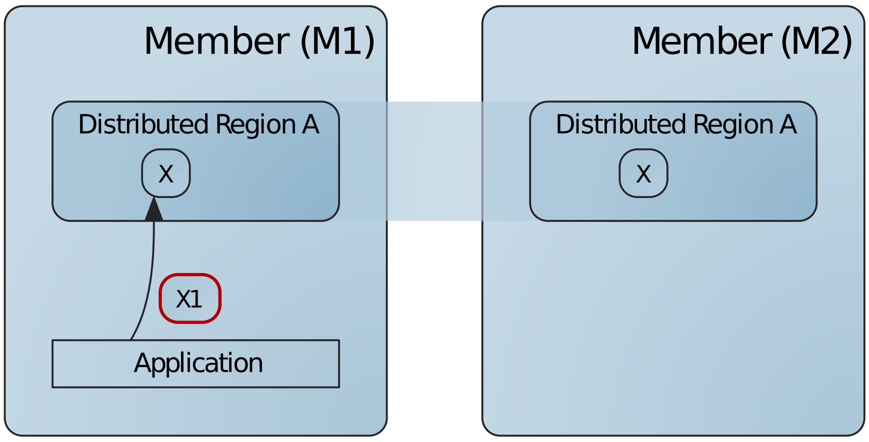
Note: The management of replicated and distributed regions supplements the general information for managing data regions provided in Basic Configuration and Programming. See also `org.apache.geode.cache.PartitionAttributes`.

A distributed region automatically sends entry value updates to remote caches and receives updates from them.

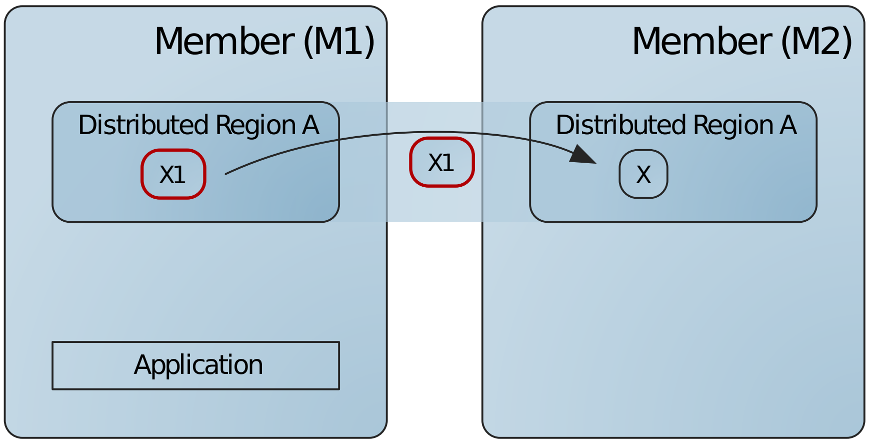
- Distributed entry updates come from the `Region` `put` and `create` operations (the creation of an entry with a non-null value is seen as an update by remote caches that already have the entry key). Entry updates are distributed selectively - only to caches where the entry key is already defined. This provides a pull model of distribution, compared to the push model that you get with replication.
- Distribution alone does not cause new entries to be copied from remote caches.
- A distributed region shares cache loader and cache writer application event handler plug-ins across the cluster.

In a distributed region, new and updated entry values are automatically distributed to remote caches that already have the entries defined.

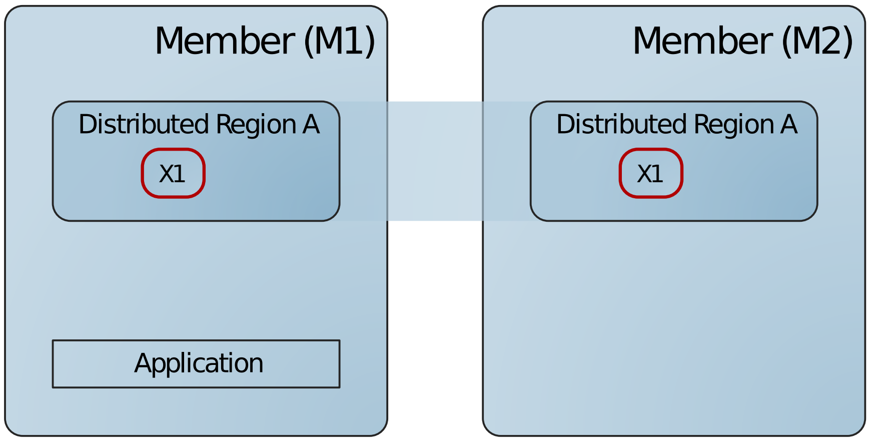
Step 1: The application updates or creates the entry. At this point, the entry in the M1 cache may not yet exist.



Step 2: The new value is automatically distributed to caches holding the entry.



Step 3: The entry's value is the same throughout the cluster.



Options for Region Distribution

You can use distribution with and without acknowledgment, or global locking for your region distribution. Regions that are configured for distribution with acknowledgment can also be configured to resolve concurrent updates consistently across all VMware GemFire members that host the region.

Each distributed region must have the same scope and concurrency checking setting throughout the cluster.

Distributed scope is provided at three levels:

- **distributed-no-ack**. Distribution operations return without waiting for a response from other caches. This scope provides the best performance and uses the least amount of overhead, but it is also most prone to having inconsistencies caused by network problems. For example, a temporary disruption of the network transport layer could cause a failure in distributing updates to a cache on a remote machine, while the local cache continues being updated.
- **distributed-ack**. Distribution waits for acknowledgment from other caches before continuing. This is slower than `distributed-no-ack`, but covers simple communication problems such as temporary network disruptions.

  In systems where there are many `distributed-no-ack` operations, it is possible for `distributed-ack` operations to take a long time to complete. The cluster has a configurable time to wait for acknowledgment to any `distributed-ack` message before sending alerts to the logs about a possible problem with the unresponsive member. No matter how long the wait, the sender keeps waiting in order to honor the distributed-ack region setting. The `gemfire.properties` attribute governing this is `ack-wait-threshold`.
- **global**. Entries and regions are automatically locked across the cluster during distribution operations. All load, create, put, invalidate, and destroy operations on the region and its entries are performed with a distributed lock. The global scope enforces strict consistency across the cluster, but it is the slowest mechanism for achieving consistency. In addition to the implicit locking performed by distribution operations, regions with global scope and their contents can be explicitly locked through the application APIs. This allows applications to perform atomic, multi-step operations on regions and region entries.

How Replication and Preloading Work

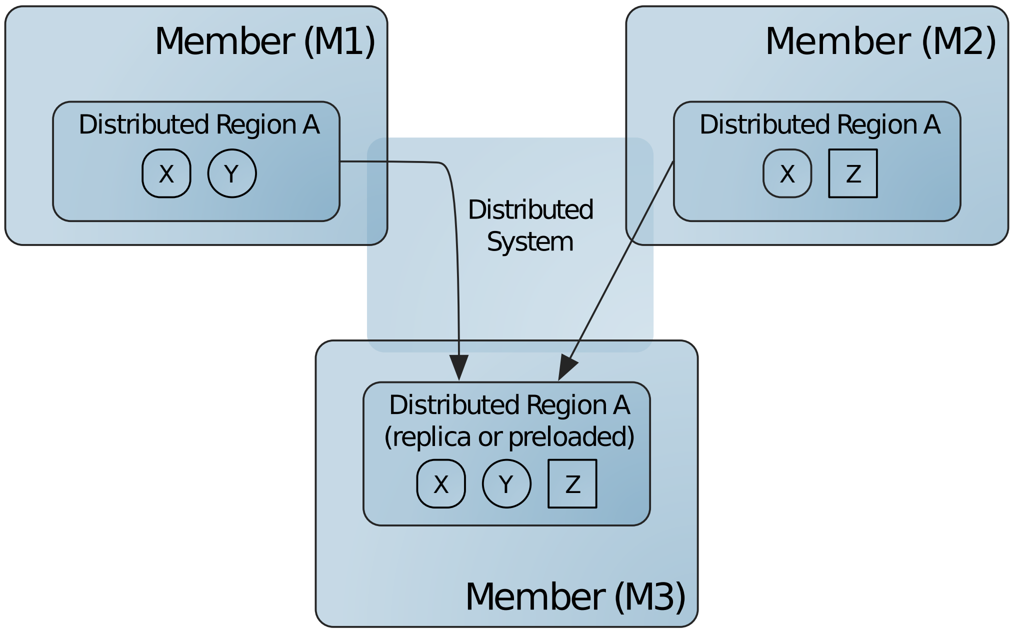
To work with replicated and preloaded regions, you should understand how their data is initialized and maintained in the cache.

Replicated and preloaded regions are configured by using one of the `REPLICATE` region shortcut settings, or by setting the region attribute `data-policy` to `replicate`, `persistent-replicate`, or `preloaded`.

Initialization of Replicated and Preloaded Regions

At region creation, the system initializes the preloaded or replicated region with the most complete and up-to-date data set it can find. The system uses these data sources to initialize the new region, following this order of preference:

1. Another replicated region that is already defined in the cluster.
2. For persistent replicate only. Disk files, followed by a union of all copies of the region in the distributed cache.
3. For preloaded region only. Another preloaded region that is already defined in the cluster.
4. The union of all copies of the region in the distributed cache.

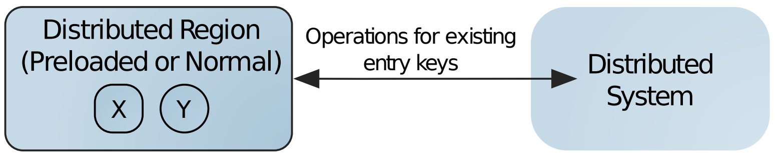


While a region is being initialized from a replicated or preloaded region, if the source region crashes, the initialization starts over.

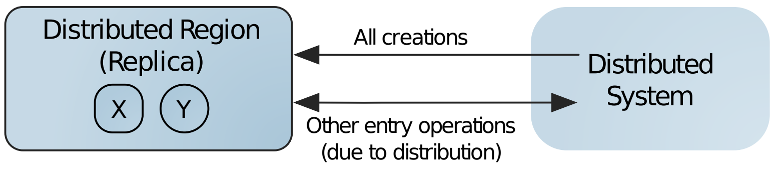
If a union of regions is used for initialization, as in the figure, and one of the individual source regions goes away during the initialization (due to cache closure, member crash, or region destruction), the new region may contain a partial data set from the crashed source region. When this happens, there is no warning logged or exception thrown. The new region still has a complete set of the remaining members' regions.

Behavior of Replicated and Preloaded Regions After Initialization

Once initialized, the preloaded region operates like the region with a `normal data-policy`, receiving distributions only for entries it has defined in the local cache.


Distributed Region (Preloaded or Normal)
X
Y
Operations for existing entry keys
Distributed System

If the region is configured as a replicated region, it receives all new creations in the distributed region from the other members. This is the push distribution model. Unlike the preloaded region, the replicated region has a contract that states it will hold all entries that are present anywhere in the distributed region.



Configure Distributed, Replicated, and Preloaded Regions

Plan the configuration and ongoing management of your distributed, replicated, and preloaded regions, and configure the regions.

Before you begin, understand Basic Configuration and Programming.

1. Choose the region shortcut setting that most closely matches your region configuration. See `org.apache.geode.cache.RegionShortcut` or Region Shortcuts. To create a replicated region, use one of the `REPLICATE` shortcut settings. To create a preloaded region, set your region `data-policy` to `preloaded`. This `cache.xml` declaration creates a replicated region:

   ```
   <region-attributes refid="REPLICATE">
   </region-attributes>
   ```

   You can also use gfsh to configure a region. For example:

   ```
   gfsh>create region --name=regionA --type=REPLICATE
   ```

   See Region Types.

2. Choose the level of distribution for your region. The region shortcuts in `RegionShortcut` for distributed regions use `distributed-ack` scope. If you need a different scope, set the `region-attributes scope` to `distributed-no-ack` or `global`.

   Example:

   ```
   <region-attributes refid="REPLICATE" scope="distributed-no-ack">
   </region-attributes>
   ```

3. If you are using the `distributed-ack` scope, optionally enable concurrency checks for the region.

   Example:

   ```
   <region-attributes refid="REPLICATE" scope="distributed-ack" concurrency-checks
   -enabled="true">
   </region-attributes>
   ```

4. If you are using `global` scope, program any explicit locking you need in addition to the automated locking provided by VMware GemFire.

Local Destroy and Invalidate in the Replicated Region

Of all the operations that affect the local cache only, only local region destroy is allowed in a replicated region. Other operations are not configurable or throw exceptions. For example, you cannot use local destroy as the expiration action on a replicated region. This is because local operations like entry invalidation and destruction remove data from the local cache only. A replicated region would no longer be complete if data were removed locally but left intact.

Locking in Global Regions

In global regions, the system locks entries and the region during updates. You can also explicitly lock the region and its entries as needed by your application. Locking includes system settings that help you optimize performance and locking behavior between your members.

In regions with global scope, locking helps ensure cache consistency.

Locking of regions and entries is done in two ways:

1. **Implicit**. VMware GemFire automatically locks global regions and their data entries during most operations. Region invalidation and destruction do not acquire locks.
2. **Explicit**. You can use the API to explicitly lock the region and its entries. Do this to guarantee atomicity in tasks with multi-step distributed operations. The `Region` methods `org.apache.geode.cache.Region.getDistributedLock` and `org.apache.geode.cache.Region.getRegionDistributedLock` return instances of `java.util.concurrent.locks.Lock` for a region and a specified key.

   Note: You must use the `Region` API to lock regions and region entries. Do not use the `DistributedLockService` in the `org.apache.geode.distributed` package. That service is available only for locking in arbitrary distributed applications. It is not compatible with the `Region` locking methods.

Lock Timeouts

Getting a lock on a region or entry is a two-step process of getting a lock instance for the entity and then using the instance to set the lock. Once you have the lock, you hold it for your operations, then release it for someone else to use. You can set limits on the time spent waiting to get a lock and the time spent holding it. Both implicit and explicit locking operations are affected by the timeouts:

- The lock timeout limits the wait to get a lock. The cache attribute `lock-timeout` governs implicit lock requests. For explicit locking, specify the wait time through your calls to the instance of `java.util.concurrent.locks.Lock` returned from the `Region` API. You can wait a specific amount of time, return immediately either with or without the lock, or wait indefinitely.

  ```
  <cache lock-timeout="60">
  </cache>
  ```

  gfsh:

  ```
  gfsh>alter runtime --lock-timeout=60
  ```

- The lock lease limits how long a lock can be held before it is automatically released. A timed lock allows the application to recover when a member fails to release an obtained lock within the lease time. For all locking, this timeout is set with the cache attribute `lock-lease`.

  ```
  <cache lock-lease="120"> </cache>
  ```

  gfsh:

  ```
  gfsh>alter runtime --lock-lease=120
  ```

Optimize Locking Performance

For each global region, one of the members with the region defined will be assigned the job of lock grantor. The lock grantor runs the lock service that receives lock requests from system members, queues them as needed, and grants them in the order received.

The lock grantor is at a slight advantage over other members as it is the only one that does not have to send a message to request a lock. The grantor's requests cost the least for the same reason. Thus, you can optimize locking in a region by assigning lock grantor status to the member that acquires the most locks. This may be the member that performs the most puts and thus requires the most implicit locks or this may be the member that performs many explicit locks.

The lock grantor is assigned as follows:

- Any member with the region defined that requests lock grantor status is assigned it. Thus at any time, the most recent member to make the request is the lock grantor.
- If no member requests lock grantor status for a region, or if the current lock grantor goes away, the system assigns a lock grantor from the members that have the region defined in their caches.

You can request lock grantor status:

1. At region creation through the `is-lock-grantor` attribute. You can retrieve this attribute through the region method, `getAttributes`, to see whether you requested to be lock grantor for the region. **Note:** The `is-lock-grantor` attribute does not change after region creation.
2. After region creation through the region `becomeLockGrantor` method. Changing lock grantors should be done with care, however, as doing so takes cycles from other operations. In particular, be careful to avoid creating a situation where you have members vying for lock grantor status.

Examples

These two examples show entry locking and unlocking. Note how the entry's `Lock` object is obtained and then its lock method invoked to actually set the lock. The example program stores the entry lock information in a hash table for future reference.

```
/* Lock a data entry */
HashMap lockedItemsMap = new HashMap();
```

```
...
  String entryKey = ...
  if (!lockedItemsMap.containsKey(entryKey))
  {
    Lock lock = this.currRegion.getDistributedLock(entryKey);
    lock.lock();
    lockedItemsMap.put(name, lock);
  }
  ...
```

```
/* Unlock a data entry */
  String entryKey = ...
  if (lockedItemsMap.containsKey(entryKey))
  {
    Lock lock = (Lock) lockedItemsMap.remove(name);
    lock.unlock();
  }
```

Consistency for Region Updates

VMware GemFire ensures that all copies of a region eventually reach a consistent state on all members and clients that host the region, including VMware GemFire members that distribute region events.

- **Consistency Checking by Region Type**

  VMware GemFire performs different consistency checks depending on the type of region you have configured.

- **Configuring Consistency Checking**

  VMware GemFire enables consistency checking by default. You cannot disable consistency checking for persistent regions. For all other regions, you can explicitly enable or disable consistency checking by setting the `concurrency-checks-enabled` region attribute in `cache.xml` to “true” or “false.”

- **Overhead for Consistency Checks**

  Consistency checking requires additional overhead for storing and distributing version and timestamp information, as well as for maintaining destroyed entries for a period of time to meet consistency requirements.

- **How Consistency Checking Works for Replicated Regions**

  Each region stores version and timestamp information for use in conflict detection. VMware GemFire members use the recorded information to detect and resolve conflicts consistently before applying a distributed update.

- **How Destroy and Clear Operations Are Resolved**

  When consistency checking is enabled for a region, a VMware GemFire member does not immediately remove an entry from the region when an application destroys the entry. Instead, the member retains the entry with its current version stamp for a period of time in order to detect possible conflicts with operations that have occurred. The retained entry is referred to as a *tombstone*. VMware GemFire retains tombstones for partitioned regions and non-replicated regions as well as for replicated regions, in order to provide consistency.

- **Transactions with Consistent Regions**

  A transaction that modifies a region having consistency checking enabled generates all necessary version information for region updates when the transaction commits.

Consistency Checking by Region Type

VMware GemFire performs different consistency checks depending on the type of region you have configured.

Partitioned Region Consistency

For a partitioned region, VMware GemFire maintains consistency by routing all updates on a given key to the VMware GemFire member that holds the primary copy of that key. That member holds a lock on the key while distributing updates to other members that host a copy of the key. Because all updates to a partitioned region are serialized on the primary VMware GemFire member, all members apply the updates in the same order and consistency is maintained at all times. See Understanding Partitioning.

Replicated Region Consistency

For a replicated region, any member that hosts the region can update a key and distribute that update to other members without locking the key. It is possible that two members can update the same key at the same time (a concurrent update). It is also possible that, due to network latency, an update in one member is distributed to other members at a later time, after those members have already applied more recent updates to the key (an out-of-order update). By default, VMware GemFire members perform conflict checking before applying region updates in order to detect and consistently resolve concurrent and out-of-order updates. Conflict checking ensures that region data eventually becomes consistent on all members that host the region. The conflict checking behavior for replicated regions is summarized as follows:

- If two members update the same key at the same time, conflict checking ensures that all members eventually apply the same value, which is the value of one of the two concurrent updates.
- If a member receives an out-of-order update (an update that is received after one or more recent updates were applied), conflict checking ensures that the out-of-order update is discarded and not applied to the cache.

How Consistency Checking Works for Replicated Regions and How Destroy and Clear Operations Are Resolved provide more details about how VMware GemFire performs conflict checking when applying an update.

Non-Replicated Regions and Client Cache Consistency

When a member receives an update for an entry in a non-replicated region and applies an update, it performs conflict checking in the same way as for a replicated region. However, if the member initiates an operation on an entry that is not present in the region, it first passes that operation to a member that hosts a replicate. The member that hosts the replica generates and provides the version information necessary for subsequent conflict checking. See How Consistency Checking Works for Replicated Regions.

Client caches also perform consistency checking in the same way when they receive an update for a region entry. However, all region operations that originate in the client cache are first passed onto an available VMware GemFire server, which generates the version information necessary for subsequent conflict checking.

Configuring Consistency Checking

VMware GemFire enables consistency checking by default. You cannot deactivate consistency checking for persistent regions. For all other regions, you can explicitly activate or deactivate consistency checking by setting the `concurrency-checks-enabled` region attribute in `cache.xml` to “true” or “false.”

All VMware GemFire members that host a region must use the same `concurrency-checks-enabled` setting for that region.

A client cache can deactivate consistency checking for a region even if server caches activate consistency checking for the same region. This configuration ensures that the client sees all events for the region, but it does not prevent the client cache region from becoming out-of-sync with the server cache.

See <region-attributes>.

Note: Regions that do not enable consistency checking remain subject to race conditions. Concurrent updates may result in one or more members having different values for the same key. Network latency can result in older updates being applied to a key after more recent updates have occurred.

Overhead for Consistency Checks

Consistency checking requires additional overhead for storing and distributing version and timestamp information, as well as for maintaining destroyed entries for a period of time to meet consistency requirements.

To provide consistency checking, each region entry uses an additional 16 bytes. When an entry is deleted, a tombstone entry of approximately 13 bytes is created and maintained until the tombstone expires or is garbage-collected in the member. (When an entry is destroyed, the member temporarily retains the entry with its current version stamp to detect possible conflicts with operations that have occurred. The retained entry is referred to as a tombstone.) See How Destroy and Clear Operations Are Resolved.

If you cannot support the additional overhead in your deployment, you can deactivate consistency checks by setting `concurrency-checks-enabled` to “false” for each region. See Consistency for Region Updates.

How Consistency Checking Works for Replicated Regions

Each region stores version and timestamp information for use in conflict detection. VMware GemFire members use the recorded information to detect and resolve conflicts consistently before applying a distributed update.

By default, each entry in a region stores the ID of the VMware GemFire member that last updated the entry, as well as a version stamp for the entry that is incremented each time an update occurs. The version information is stored in each local entry, and the version stamp is distributed to other VMware GemFire members when the local entry is updated.

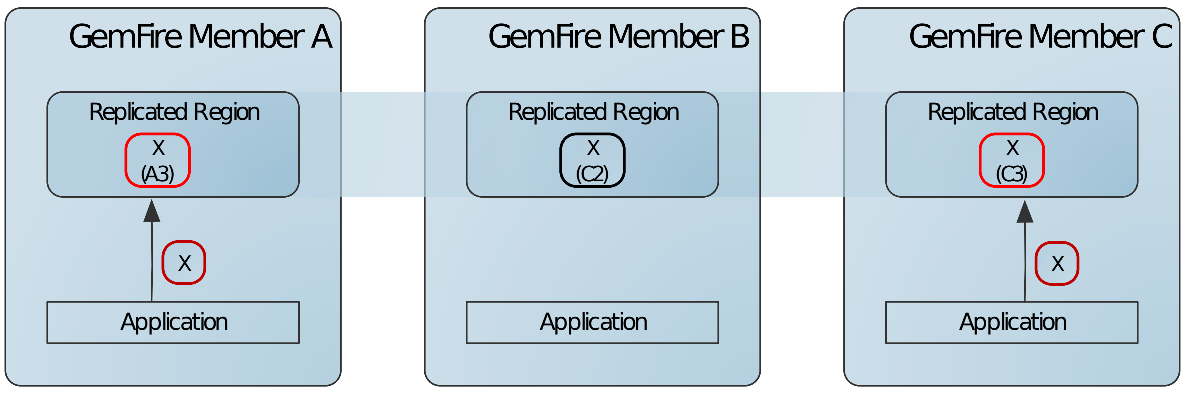
A VMware GemFire member or client that receives an update message first compares the update version stamp with the version stamp recorded in its local cache. If the update version stamp is larger, it represents a newer version of the entry, so the receiving member applies the update locally and updates the version information. A smaller update version stamp indicates an out-of-order update, which is discarded.

An identical version stamp indicates that multiple VMware GemFire members updated the same entry at the same time. To resolve a concurrent update, a VMware GemFire member always applies (or keeps) the region entry that has the highest membership ID; the region entry having the lower membership ID is discarded.

Note: When a VMware GemFire member discards an update message (either for an out-of-order update or when resolving a concurrent update), it does not pass the discarded event to an event listener for the region. You can track the number of discarded updates for each member using the `conflatedEvents` statistic. See VMware GemFire Statistics List. Some members may discard an update while other members apply the update, depending on the order in which each member receives the update. For this reason, the `conflatedEvents` statistic differs for each VMware GemFire member. The example below describes this behavior in more detail.

The following example shows how a concurrent update is handled in a cluster of three VMware GemFire members. Assume that Members A, B, and C have membership IDs of 1, 2, and 3, respectively. Each member currently stores an entry, X, in their caches at version C2 (the entry was last updated by member C):

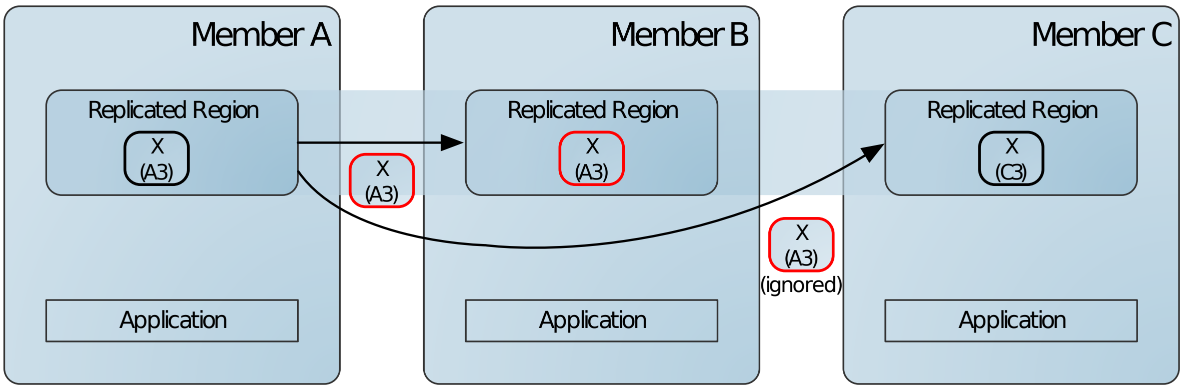
Step 1: An application updates entry X on VMware GemFire member A at the same time another application updates entry X on member C. Each member increments the version stamp for the entry and records the version stamp with their member ID in their local caches. In this case the entry was originally at version C2, so each member updates the version to 3 (A3 and C3, respectively) in their local caches.



Step 2: Member A distributes its update message to members B and C.

Member B compares the update version stamp (3) to its recorded version stamp (2) and applies the update to its local cache as version A3. In this member, the update is applied for the time being, and passed on to configured event listeners.

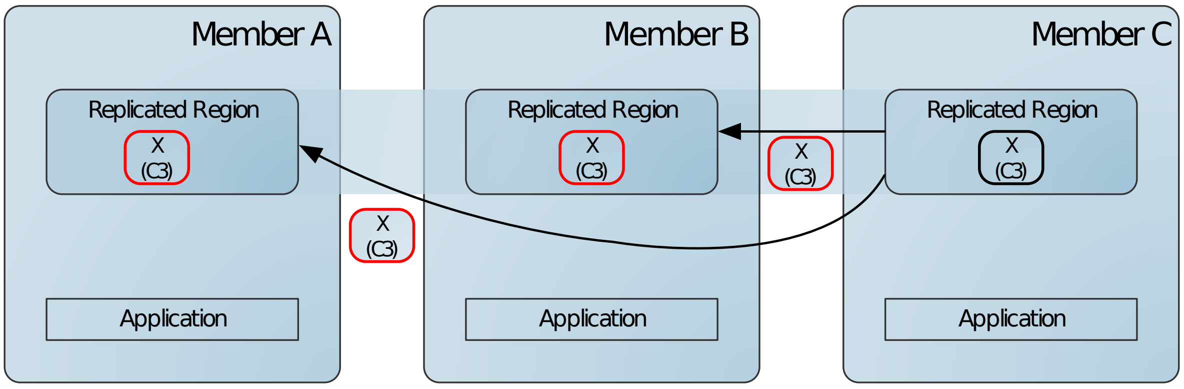
Member C compares the update version stamp (3) to its recorded version stamp (3) and identifies a concurrent update. To resolve the conflict, member C next compares the membership ID of the update to the membership ID stored in its local cache. Because the distributed system ID the update (A3) is lower than the ID stored in the cache (C3), member C discards the update (and increments the `conflatedEvents` statistic).



Step 3: Member C distributes the update message to members A and B.

Members A and B compare the update version stamp (3) to their recorded version stamps (3) and identify the concurrent update. To resolve the conflict, both members compare the membership ID of the update with the membership ID stored in their local caches. Because the distributed system ID of A in the cache value is less than the ID of C in the update, both members record the update C3 in their local caches, overwriting the previous value.

At this point, all members that host the region have achieved a consistent state for the concurrent updates on members A and C.



How Destroy and Clear Operations Are Resolved

When consistency checking is enabled for a region, a VMware GemFire member does not immediately remove an entry from the region when an application destroys the entry. Instead, the member retains the entry with its current version stamp for a period of time in order to detect possible conflicts with operations that have occurred. The retained entry is referred to as a *tombstone*. VMware GemFire retains tombstones for partitioned regions and non-replicated regions as well as for replicated regions, in order to provide consistency.

A tombstone in a client cache or a non-replicated region expires after 8 minutes, at which point the tombstone is immediately removed from the cache.

A tombstone for a replicated or partitioned region expires after 10 minutes. Expired tombstones are eligible for garbage collection by the VMware GemFire member. Garbage collection is automatically triggered after 100,000 tombstones of any type have timed out in the local VMware GemFire member. You can optionally set the `gemfire.tombstone-gc-threshold` property to a value smaller than 100000 to perform garbage collection more frequently.

Note: To avoid out-of-memory errors, a VMware GemFire member also initiates garbage collection for tombstones when the amount of free memory drops below 30 percent of total memory.

You can monitor the total number of tombstones in a cache using the `tombstoneCount` statistic in `CachePerfStats`. The `tombstoneGCCount` statistic records the total number of tombstone garbage collection cycles that a member has performed. `replicatedTombstonesSize` and `nonReplicatedTombstonesSize` show the approximate number of bytes that are currently consumed by tombstones in replicated or partitioned regions, and in non-replicated regions, respectively. See VMware GemFire Statistics List.

About Region.clear() Operations

Region entry version stamps and tombstones ensure consistency only when individual entries are destroyed. A `Region.clear()` operation, however, operates on all entries in a region at once. To provide consistency for `Region.clear()` operations, VMware GemFire obtains a distributed read/write lock for the region, which blocks all concurrent updates to the region. Any updates that were initiated before the clear operation are allowed to complete before the region is cleared.

Transactions with Consistent Regions

A transaction that modifies a region having consistency checking enabled generates all necessary version information for region updates when the transaction commits.

If a transaction modifies a normal, preloaded or empty region, the transaction is first delegated to a VMware GemFire member that holds a replicate for the region. This behavior is similar to the transactional behavior for partitioned regions, where the partitioned region transaction is forwarded to a member that hosts the primary for the partitioned region update.

The limitation for transactions on normal, preloaded or or empty regions is that, when consistency checking is enabled, a transaction cannot perform a `localDestroy` or `localInvalidate` operation against the region. VMware GemFire throws an `UnsupportedOperationInTransactionException` exception in such cases. An application should use a `Destroy` or `Invalidate` operation in place of a `localDestroy` or `localInvalidate` when consistency checks are enabled.

How Consistency Is Achieved in WAN Deployments

When two or more VMware GemFire systems are configured to distribute events over a WAN, each system performs local consistency checking before it distributes an event to a configured gateway sender. Discarded events are not distributed across the WAN.

Regions can also be configured to distribute updates to other VMware GemFire clusters over a WAN. With a distributed WAN configuration, multiple gateway senders asynchronously queue and send region updates to another VMware GemFire cluster. It is possible for multiple sites to send updates to the same region entry at the same time. It is also possible that, due to a slow WAN connection, a cluster might receive region updates after a considerable delay, and after it has applied more recent updates to a region. To ensure that WAN-replicated regions eventually reach a consistent state, VMware GemFire first ensures that each cluster performs consistency checking to regions before queuing updates to a gateway sender for WAN distribution. In order words, region conflicts are first detected and resolved in the local cluster, using the techniques described in the previous sections.

When a VMware GemFire cluster in a WAN configuration receives a distributed update, conflict checking is performed to ensure that all sites apply updates in the same way. This ensures that regions eventually reach a consistent state across all VMware GemFire clusters. The default conflict checking behavior for WAN-replicated regions is summarized as follows:

- If an update is received from the same VMware GemFire cluster that last updated the region entry, then there is no conflict and the update is applied.
- If an update is received from a different VMware GemFire cluster than the one that last updated the region entry, then a potential conflict exists. A cluster applies the update only when the update has a timestamp that is later than the timestamp currently recorded in the cache.

Note: If you use the default conflict checking feature for WAN deployments, you must ensure that all VMware GemFire members in all clusters synchronize their system clocks. For example, use a common NTP server for all VMware GemFire members that participate in a WAN deployment.

As an alternative to the default conflict checking behavior for WAN deployments, you can develop and deploy a custom conflict resolver for handling region events that are distributed over a WAN. Using a custom resolver enables you to handle conflicts using criteria other than, or in addition to, timestamp information. For example, you might always prioritize updates that originate from a particular site, given that the timestamp value is within a certain range.

When a gateway sender distributes an event to another VMware GemFire site, it adds the distributed system ID of the local cluster, as well as a timestamp for the event. In a default configuration, the cluster that receives the event examines the timestamp to determine whether or not the event should be applied. If the timestamp of the update is earlier than the local timestamp, the cluster discards the event. If the timestamp is the same as the local timestamp, then the entry having the highest distributed system ID is applied (or kept).

You can override the default consistency checking for WAN events by installing a conflict resolver plug-in for the region. If a conflict resolver is installed, then any event that can potentially cause a conflict (any event that originated from a different distributed system ID than the ID that last modified the entry) is delivered to the conflict resolver. The resolver plug-in then makes the sole determination for which update to apply or keep.

See “Implementing a GatewayConflictResolver” under Resolving Conflicting Events to configure a custom resolver.

General Region Data Management

For all regions, you have options to control memory use, back up your data to disk, and discard stale data from your cache.

- **Persistence and Overflow**

  You can persist data on disk for backup purposes and overflow it to disk to free up memory without completely removing the data from your cache.

- **Eviction**

  Use eviction to control data region size. Eviction actions are triggered by space-based thresholds.

- **Expiration**

  Use expiration to keep data current and to reduce region size by removing stale entries. Expiration actions are triggered by time-based thresholds.

- **Keeping the Cache in Sync with Outside Data Sources**

  Keep your distributed cache in sync with an outside data source by programming and installing application plug-ins for your region.

Persistence and Overflow

You can persist data on disk for backup purposes and overflow it to disk to free up memory without completely removing the data from your cache.

Note: This supplements the general steps for managing data regions provided in Basic Configuration and Programming.

All disk storage uses VMware GemFire Disk Storage.

- **How Persistence and Overflow Work**

  To use VMware GemFire persistence and overflow, you should understand how they work with your data.

- **Configure Region Persistence and Overflow**

  Plan persistence and overflow for your data regions and configure them accordingly.

- **Overflow Configuration Examples**

  The `cache.xml` examples show configuration of region and server subscription queue overflows.

How Persistence and Overflow Work

To use VMware GemFire persistence and overflow, you should understand how they work with your data.

VMware GemFire persists and overflows several types of data. You can persist or overflow the application data in your regions. In addition, VMware GemFire persists and overflows messaging queues, to manage memory consumption and provide high availability.

Persistent data outlives the member where the region resides and can be used to initialize the region at creation. Overflow acts only as an extension of the region in memory.

The data is written to disk according to the configuration of VMware GemFire disk stores. For any disk option, you can specify the name of the disk store to use or use the VMware GemFire default disk store. See Disk Storage.

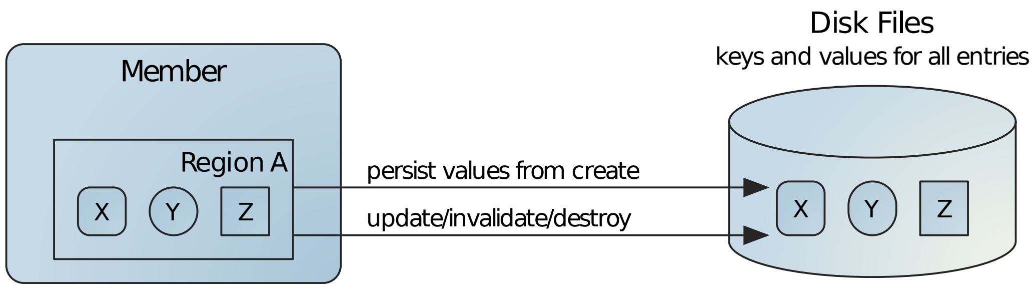
How Data Is Persisted and Overflowed

For persistence, the entry keys and values are copied to disk. For overflow, only the entry values are copied. Other data, such as statistics and user attributes, are retained in memory only.

- Data regions are overflowed to disk by least recently used (LRU) entries because those entries are deemed of least interest to the application and therefore less likely to be accessed.
- Server subscription queues overflow most recently used (MRU) entries. These are the messages that are at the end of the queue and so are last in line to be sent to the client.

Persistence

Persistence provides a disk backup of region entry data. The keys and values of all entries are saved to disk, like having a replica of the region on disk. Region entry operations such as put and destroy are carried out in memory and on disk.

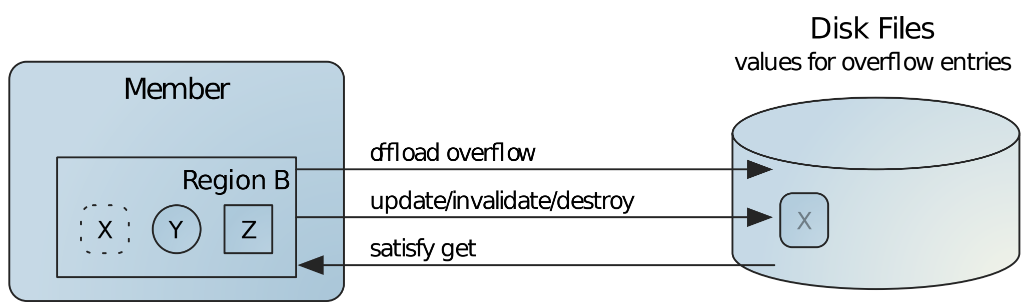

Member
Region A
X
Y
Z
persist values from create
update/invalidate/destroy
Disk Files
keys and values for all entries
X
Y
Z

When the member stops for any reason, the region data on disk remains. In partitioned regions, where data buckets are divided among members, this can result in some data only on disk and some on disk and in memory. The disk data can be used at member startup to populate the same region.

Overflow

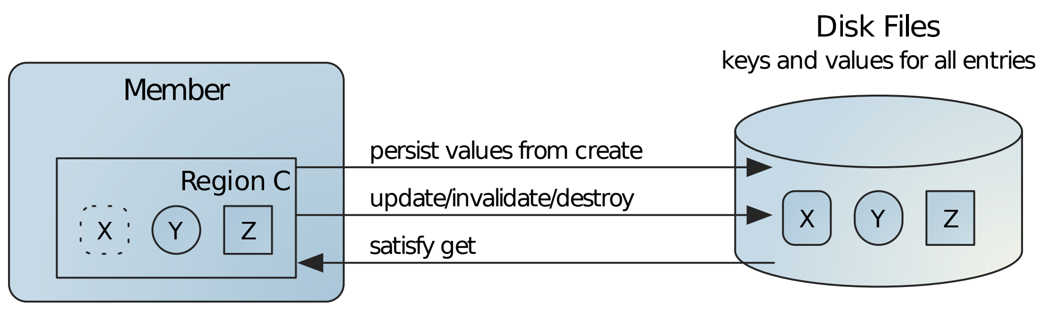
Overflow limits region size in memory by moving the values of least recently used (LRU) entries to disk. Overflow basically uses disk as a swap space for entry values. If an entry is requested whose value is only on disk, the value is copied back up into memory, possibly causing the value of a different LRU entry to be moved to disk. As with persisted entries, overflowed entries are maintained on disk just as they are in memory.

In this figure, the value of entry X has been moved to disk to make space in memory. The key for X remains in memory. From the distributed system perspective, the value on disk is as much a part of the region as the data in memory.



Persistence and Overflow Together

Used together, persistence and overflow keep all entry keys and values on disk and only the most active entry values in memory. The removal of an entry value from memory due to overflow has no effect on the disk copy as all entries are already on disk.


Disk Files
keys and values for all entries
Member
Region C
X
Y
Z
persist values from create
update/invalidate/destroy
satisfy get
X
Y
Z

Persistence and Multi-Site Configurations

Multi-site gateway sender queues overflow most recently used (MRU) entries. These are the messages that are at the end of the queue and so are last in line to be sent to the remote site. You can also configure gateway sender queues to persist for high availability.

Configure Region Persistence and Overflow

Plan persistence and overflow for your data regions and configure them accordingly.

Use the following steps to configure your data regions for persistence and overflow:

1. Configure your disk stores as needed. See Designing and Configuring Disk Stores. The cache disk store defines where and how the data is written to disk.

```
<disk-store name="myPersistentStore" . . . >
<disk-store name="myOverflowStore" . . . >
```

2. Specify the persistence and overflow criteria for the region. If you are not using the default disk store, provide the disk store name in your region attributes configuration. To write asynchronously to disk, specify `disk-synchronous="false"`.

   - For overflow, specify the overflow criteria in the region's `eviction-attributes` and name the disk store to use.

Example:

```
<region name="overflowRegion" . . . >
  <region-attributes disk-store-name="myOverflowStore" disk-synchronous
="true">
    <eviction-attributes>
      <!-- Overflow to disk when 100 megabytes of data reside in the
           region -->
      <lru-memory-size maximum="100" action="overflow-to-disk"/>
    </eviction-attributes>
  </region-attributes>
</region>
```

gfsh:

You cannot configure `lru-memory-size` using gfsh.

- For persistence, set the `data-policy` to `persistent-replicate` and name the disk store to use.

  Example:

```
<region name="partitioned_region" refid="PARTITION_PERSISTENT">
  <region-attributes disk-store-name="myPersistentStore">
    . . .
  </region-attributes>
</region>
```

When you start your members, overflow and persistence will be done automatically, with the disk stores and disk write behaviors.

Note: You can also configure Regions and Disk Stores using the gfsh command-line interface. See Region Commands and Disk Store Commands.

Related Topics
`org.apache.geode.cache.RegionAttributes` for data region persistence information
`org.apache.geode.cache.EvictionAttributes` for data region overflow information
`org.apache.geode.cache.server.ClientSubscriptionConfig`

Overflow Configuration Examples

The `cache.xml` examples show configuration of region and server subscription queue overflows.

Configure overflow criteria based on one of these factors:

- Entry count
- Absolute memory consumption
- Memory consumption as a percentage of the application heap (not available for server subscription queues)

Configuration of region overflow:

```
<!-- Overflow when the region goes over 10000 entries -->
<region-attributes>
  <eviction-attributes>
    <lru-entry-count maximum="10000" action="overflow-to-disk"/>
  </eviction-attributes>
</region-attributes>
```

Configuration of server's client subscription queue overflow:

```
<!-- Overflow the server's subscription queues when the queues reach 1 Mb of memory --
>
<cache>
  <cache-server>
    <client-subscription eviction-policy="mem" capacity="1"/>
  </cache-server>
</cache>
```

Eviction

Use eviction to control data region size. Eviction actions are triggered by space-based thresholds.

- **How Eviction Works**

  Eviction settings cause VMware GemFire to work to keep a region's resource use under a specified level by removing least recently used (LRU) entries to make way for new entries.

- **Configure Data Eviction**

  Configure a region's `eviction-attributes` settings to keep your region within a specified limit.

How Eviction Works

Eviction keeps a region's resource use under a specified level by removing least recently used (LRU) entries to make way for new entries. You can choose whether expired entries are overflowed to disk or destroyed. See Persistence and Overflow.

Eviction is triggered when a size-based threshold is exceeded. A region's eviction threshold can be based on:

- entry count
- absolute memory usage
- percentage of available heap

These eviction algorithms are mutually exclusive; only one can be in effect for a given region.

When VMware GemFire determines that adding or updating an entry would take the region over the specified level, it overflows or removes enough older entries to make room. For entry count eviction, this means a one-to-one trade of an older entry for the newer one. For the memory settings, the number of older entries that need to be removed to make space depends on the sizes of the older and newer entries.

For efficiency, the selection of items for removal is not strictly LRU, but does choose eviction candidates from among the region's oldest entries. As a result, eviction may leave older entries for the region in the local data store.

Eviction Actions

VMware GemFire provides the following eviction actions:

- **local destroy** - Removes the entry from the local cache, but does not distribute the removal operation to remote members. This action can be applied to an entry in a partitioned region, but is not recommended if redundancy is enabled (redundant-copies > 0), as it introduces inconsistencies between the redundant buckets. When applied to an entry in a replicated region, VMware GemFire silently changes the region type to "preloaded" to accommodate the local modification.
- **overflow to disk** - The entry's value is overflowed to disk and set to null in memory. The entry's key is retained in the cache. This is the only eviction action fully supported for partitioned regions.

Eviction in Partitioned Regions

In partitioned regions, VMware GemFire removes the oldest entry it can find *in the bucket where the new entry operation is being performed*. VMware GemFire maintains LRU entry information on a bucket-by-bucket basis, as the cost of maintaining information across the partitioned region would slow the system's performance.

- For memory and entry count eviction, LRU eviction is done in the bucket where the new entry operation is being performed until the overall size of the combined buckets in the member has dropped enough to perform the operation without going over the limit.
- For heap eviction, each partitioned region bucket is treated as if it were a separate region, with each eviction action only considering the LRU for the bucket, and not the partitioned region as a whole.

Configure Data Eviction

Configure a region's `eviction-attributes` settings to keep your region within a specified limit.

Configure data eviction as follows. You do not need to perform these steps in the sequence shown.

1. Decide whether to evict based on:
    - Entry count (useful if your entry sizes are relatively uniform).
    - Total bytes used. In partitioned regions, this is set using `local-max-memory`. In non-partitioned regions, it is set in `eviction-attributes`.
    - Percentage of application heap used. This uses the VMware GemFire resource manager. When the manager determines that eviction is required, the manager orders the eviction controller to start evicting from all regions where the eviction algorithm is set to `lru-heap-percentage`. Eviction continues until the manager calls a halt. VMware GemFire evicts the least recently used entry hosted by the member for the region. See Managing Heap and Off-heap Memory.
2. Decide what action to take when the limit is reached:

    - Locally destroy the entry.
    - Overflow the entry data to disk. See Persistence and Overflow.
3. Decide the maximum amount of data to allow in the member for the eviction measurement indicated. This is the maximum for all storage for the region in the member. For partitioned regions, this is the total for all buckets stored in the member for the region, including any secondary buckets used for redundancy.
4. Decide whether to program a custom sizer for your region. If you are able to provide such a class, it might be faster than the standard sizing done by VMware GemFire. Your custom class must follow the guidelines for defining custom classes and, additionally, must implement `org.apache.geode.cache.util.ObjectSizer`. See Requirements for Using Custom Classes in Data Caching.

Examples:

Set an LRU memory eviction threshold of 1000 MB. Use a custom class for measuring the size of each object in the region:

```
gfsh>create region --name=myRegion --type=REPLICATE --eviction-max-memory=1000 \
--eviction-action=overflow-to-disk --eviction-object-sizer=com.myLib.MySizer
```

Create an eviction threshold on a partitioned region with a maximum entry count of 512:

```
gfsh>create region --name=myRegion --type=PARTITION --eviction-entry-count=512 \
--eviction-action=overflow-to-disk
```

To configure a partitioned region for heap LRU eviction, first configure the resource manager on server startup, then create a region with eviction enabled:

```
gfsh>start server --name=Server1 --eviction-heap-percentage=80
...
gfsh>create region --name=myRegion --type=PARTITION --eviction-action=overflow-to-disk
```

Expiration

Use expiration to keep data current and to reduce region size by removing stale entries. Expiration actions are triggered by time-based thresholds.

- **How Expiration Works**

  Expiration removes old entries and entries that you are not using. You can choose whether expired entries are invalidated or destroyed.

- **Configure Data Expiration**

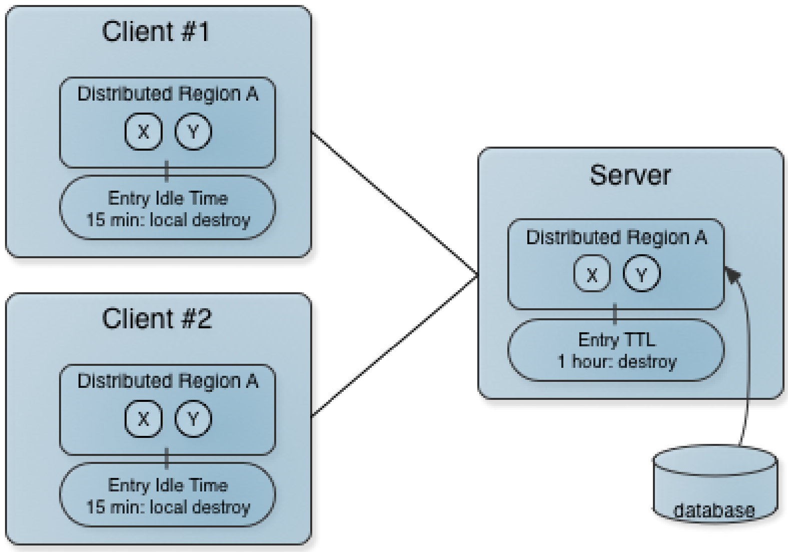
  Configure the type of expiration and the expiration action to use.

How Expiration Works

Expiration keeps a region's data fresh by removing old entries and entries that you are not using. You can choose whether expired entries are invalidated or destroyed.

Expiration activities in distributed regions can be distributed or local. Thus, one cache could control expiration for a number of caches in the system.

This figure shows two basic expiration settings for a client/server system. The server (on the right) populates the region from a database and the data is automatically distributed throughout the system. The data is valid for only one hour, so the server performs a distributed destroy on entries that are an hour old. The client applications are consumers. The clients free up space in their caches by removing their local copies of the entries for which there is no local interest (idle-time expiration). Requests for entries that have expired on the clients will be forwarded to the server.



Expiration Types

VMware GemFire provides two types of expiration, each triggered by a time-based threshold. These can co-exist; they are not mutually exclusive.

- **Time to live (TTL)**. The amount of time, in seconds, the object may remain in the cache after the last creation or update. For entries, the counter is set to zero for create and put operations. Region counters are reset when the region is created and when an entry has its counter reset. The TTL expiration attributes are `region-time-to-live` and `entry-time-to-live`.
- **Idle timeout**. The amount of time, in seconds, the object may remain in the cache after the last access. The idle timeout counter for an object is reset any time its TTL counter is reset. In addition, an entry's idle timeout counter is reset any time the entry is accessed through a get operation or a netSearch . The idle timeout counter for a region is reset whenever the idle timeout is reset for one of its entries. Idle timeout expiration attributes are: `region-idle-time` and `entry-idle-time`.

Expiration Actions

VMware GemFire provides the following expiration actions:

- **invalidate (default)** - The data item's value is deleted, but the key remains in the cache. Applies to all distributed members in which the data item is replicated.

- **destroy** - The data item's key and value are both deleted. Applies to all distributed members in which the data item is replicated.
- **local invalidate** - Deletes the data item's value. Applies only to the local member.
- **local destroy** - Deletes the data item's key and value. Applies only to the local member.

You cannot use `local-destroy` or `local-invalidate` expiration actions in replicated or partitioned regions. You can use the local options only on distributed regions with a data-policy of empty, normal or preloaded.

Entry Expiration in Replicated Regions and Partitioned Regions

In replicated regions, entry updates are performed in the most convenient available copy of the data, then replicated to the other members, resetting their last-updated statistics to the same time. In partitioned regions, entry updates are always done in the primary copy, resetting the primary copy's last-updated and last-accessed statistics, then the secondary copies are updated to match.

In both replicated and partitioned regions, entry retrieval uses the most convenient available copy of the data, which may be any of the distributed copies. Retrievals are not propagated to other members. Differences in last-access times are reconciled when the data item is considered for expiration.

Expiration can be triggered in any copy of a replicated region, if the time elapsed since the last update or read access exceeds the established threshold. Expiration in partitioned regions is executed in the primary copy, based on the primary's last-accessed and last-updated statistics. In both cases, the expiration mechanism checks the last-accessed dates of all copies of the data item and updates the last-access date of all copies to the most recent last-accessed date. Then, if the elapsed time still puts the data item over the expiration threshold, the item is deleted in accordance with the expiration action specified for the region.

Interaction Between Expiration Settings and netSearch

Before `netSearch` retrieves an entry value from a remote cache, it validates the *remote* entry's statistics against the *local* region's expiration settings. Entries that would have already expired in the local cache are passed over. Once validated, the entry is brought into the local cache and the local access and update statistics are updated for the local copy. The last-accessed time is reset and the last-modified time is updated to the time in the remote cache, with corrections made for system clock differences. Thus the local entry is assigned the true last time the entry was modified in the cluster. The `netSearch` operation has no effect on the expiration counters in remote caches.

The `netSearch` method operates only on distributed regions with a data-policy of empty, normal and preloaded.

Configure Data Expiration

Configure the type of expiration and the expiration action to use.

- Expiration actions require setting the region attribute of `statistics-enabled` to `true`. This can be done in the region element of a `cache.xml` file, the `gfsh` command line, or through the API.

- Set the expiration attributes by expiration type, with the max times and expiration actions. See the region attributes listings for `entry-time-to-live`, `entry-idle-time`, `region-time-to-live`, and `region-idle-time` in <region-attributes>.

The statistics used for expiration are available directly to the application through the `CacheStatistics` object returned by the `Region` and `Region.Entry` `getStatistics` methods. The `CacheStatistics` object also provides a method for resetting the statistics counters.

For partitioned regions:

- On a partitioned region, expiration is supported only for the region's entries, not for the region itself. Region-wide expiration attributes, such as `region-time-to-live` and `region-idle-time` do not apply to the data items in partitioned regions.
- To ensure reliable read behavior when working with partitioned regions, use the `entry-time-to-live` attribute, not the `entry-idle-time` attribute.
- You cannot use `local-destroy` or `local-invalidate` expiration actions in partitioned regions.

Replicated regions example:

```
// Setting standard expiration on an entry
<region-attributes statistics-enabled="true">
  <entry-idle-time>
    <expiration-attributes timeout="60" action="local-invalidate"/>
  </entry-idle-time>
</region-attributes>
```

- Override the region-wide settings for specific entries, if required by your application. To do this:
  1. Program a custom expiration class that implements `org.apache.geode.cache.CustomExpiry`. Example:

     ```
     // Custom expiration class
     // Use the key for a region entry to set entry-specific expiration timeou
     ts of
     //   10 seconds for even-numbered keys with a DESTROY action on the expir
     ed entries
     //   Leave the default region setting for all odd-numbered keys.
     public class MyClass implements CustomExpiry, Declarable
     {
         private static final ExpirationAttributes CUSTOM_EXPIRY =
                 new ExpirationAttributes(10, ExpirationAction.DESTROY);
         public ExpirationAttributes getExpiry(Entry entry)
         {
             int key = (Integer)entry.getKey();
             return key % 2 == 0 ? CUSTOM_EXPIRY : null;
         }
     }
     ```

  2. Define the class inside the expiration attributes settings for the region. Example:

```
<!-- Set default entry idle timeout expiration for the region -->
<!-- Pass entries to custom expiry class for expiration overrides -->
<region-attributes statistics-enabled="true">
    <entry-idle-time>
        <expiration-attributes timeout="60">
            <custom-expiry>
                <class-name>com.company.mypackage.MyClass</class-name>
            </custom-expiry>
        </expiration-attributes>
    </entry-idle-time>
</region-attributes>
```

The gfsh equivalent of the above XML is:

```
gfsh> create region --name=region1 --type=REPLICATE --enable-statistics \
--entry-idle-time-expiration=60 --entry-idle-time-custom-expiry=com.company.myp
ackage.MyClass
```

- When the primary expires entries, it requests last-accessed statistics from the secondaries. The primary adopts the most recent access time and reschedules the expiration, if warranted. This is done only for distributed expiration actions, and applies to both partitioned and replicated regions.

You can also configure regions using the gfsh command-line interface. See Region Commands.

Configuring the Number of Threads for Expiration

You can use the `gemfire.EXPIRY_THREADS` system property to increase the number of threads that handle expiration. By default, one thread handles expiration, and it is possible for the thread to become overloaded when entries expire faster than the thread can expire them. If a single thread is handling too many expirations, it can result in an OOME. Set the gemfire.EXPIRY_THREADS system property to the desired number when starting the cache server.

Keeping the Cache in Sync with Outside Data Sources

Keep your distributed cache in sync with an outside data source by programming and installing application plug-ins for your region.

- **Overview of Outside Data Sources**

  VMware GemFire has application plug-ins to read data into the cache and write it out.

- **Configuring Database Connections Using JNDI**.

  Use JNDI to maintain a connection pool that includes outside data sources.

- **How Data Loaders Work**

  By default, a region has no data loader defined. Plug an application-defined loader into any region by setting the region attribute cache-loader on the members that host data for the region.

- **Implement a Data Loader**

  Program a data loader and configure your region to use it.

Overview of Outside Data Sources

VMware GemFire has application plug-ins to read data into the cache and write it out.

The application plug-ins:

1. Load data on cache misses using an implementation of a `org.apache.geode.cache.CacheLoader`. The `CacheLoader.load` method is called when the `get` operation can't find the value in the cache. The value returned from the loader is put into the cache and returned to the `get` operation. You might use this in conjunction with data expiration to get rid of old data, and your other data loading applications, which might be prompted by events in the outside data source. See Configure Data Expiration.
2. Write data out to the data source using the cache event handlers, `CacheWriter` and `CacheListener`. For implementation details, see Implementing Cache Event Handlers.
    - `CacheWriter` is run synchronously. Before performing any operation on a region entry, if any cache writers are defined for the region in the cluster, the system invokes the most convenient writer. In partitioned and distributed regions, cache writers are usually defined in only a subset of the caches holding the region - often in only one cache. The cache writer can cancel the region entry operation.
    - `CacheListener` is run synchronously after the cache is updated. This listener works only on local cache events, so install your listener in every cache where you want it to handle events. You can install multiple cache listeners in any of your caches.

In addition to using application plug-ins, you can also configure external JNDI database sources in your cache.xml and use these data sources in transactions. See Configuring Database Connections Using JNDI for more information.

Configuring Database Connections Using JNDI

To connect to external databases, for example when using JTA transactions, you can configure database JNDI data sources in `cache.xml`. The `DataSource` object points to either a JDBC connection or, more commonly, a JDBC connection pool. The connection pool is usually preferred, because a program can use and reuse a connection as long as necessary and then free it for another thread to use.

The following list shows `DataSource` connection types used in JTA transactions:

- **XAPooledDataSource**. Pooled SQL connections.
- **ManagedDataSource**. JNDI binding type for the J2EE Connector Architecture (JCA) ManagedConnectionFactory.
- **PooledDataSource**. Pooled SQL connections.
- **SimpleDataSource**. Single SQL connection. No pooling of SQL connections is done. Connections are generated on the fly and cannot be reused.

The `jndi-name` attribute of the `jndi-binding` element is the key binding parameter. If the value of `jndi-name` is a DataSource, it is bound as `java:/`*myDatabase*, where *myDatabase* is the name you assign to your data source. If the data source cannot be bound to JNDI at runtime, VMware GemFire logs a warning. For information on the `DataSource` interface, see: http://docs.oracle.com/javase/8/docs/api/javax/sql/DataSource.html

VMware GemFire supports JDBC 2.0 and 3.0.

Note: Include any data source JAR files in your CLASSPATH.

Example DataSource Configurations in cache.xml

The following sections show example `cache.xml` files configured for each of the `DataSource` connection types.

XAPooledDataSource cache.xml Example (Derby)

The example shows a `cache.xml` file configured for a pool of `XAPooledDataSource` connections connected to the data resource `newDB`.

The log-in and blocking timeouts are set lower than the defaults. The connection information, including `user-name` and `password`, is set in the `cache.xml` file, instead of waiting until connection time. The password is not encrypted.

When specifying the configuration properties for JCA-implemented database drivers that support XA transactions (in other words, **XAPooledDataSource**), you must use configuration properties to define the datasource connection instead of the `connection-url` attribute of the `<jndi-binding>` element. Configuration properties differ depending on your database vendor. Specify JNDI binding properties through the `config-property` tag, as shown in this example. You can add as many `config-property` tags as required.

```
<?xml version="1.0" encoding="UTF-8"?>
<cache
    xmlns="http://geode.apache.org/schema/cache"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://geode.apache.org/schema/cache http://geode.apache.org/s
chema/cache/cache-1.0.xsd"
    version="1.0"
    lock-lease="120" lock-timeout="60" search-timeout="300">
   <region name="root">
      <region-attributes scope="distributed-no-ack" data-policy="cached" initial-capac
ity="16"
load-factor="0.75" concurrency-level="16" statistics-enabled="true">
    . . .
   </region>
   <jndi-bindings>
      <jndi-binding type="XAPooledDataSource"
    jndi-name="newDB2trans"
    init-pool-size="20"
    max-pool-size="100"
    idle-timeout-seconds="20"
    blocking-timeout-seconds="5"
    login-timeout-seconds="10"
    xa-datasource-class="org.apache.derby.jdbc.EmbeddedXADataSource"
    user-name="mitul"
    password="thecleartextpassword">
         <config-property>
          <config-property-name>Description</config-property-name>
          <config-property-type>java.lang.String</config-property-type>
          <config-property-value>pooled_transact</config-property-value>
       </config-property>
```

```
          <config-property>
             <config-property-name>DatabaseName</config-property-name>
             <config-property-type>java.lang.String</config-property-type>
             <config-property-value>newDB</config-property-value>
          </config-property>
          <config-property>
             <config-property-name>CreateDatabase</config-property-name>
             <config-property-type>java.lang.String</config-property-type>
             <config-property-value>create</config-property-value>
          </config-property>
       . . .
      </jndi-binding>
   </jndi-bindings>
</cache>
```

JNDI Binding Configuration Properties for Different XAPooledDataSource Connections

The following are some example data source configurations for different databases. Consult your vendor database's documentation for additional details.

MySQL

```
...
<jndi-bindings>
   <jndi-binding type="XAPooledDataSource"
    ...
    xa-datasource-class="com.mysql.jdbc.jdbc2.optional.MysqlXADataSource">
    <config-property>
    <config-property-name>URL</config-property-name>
    <config-property-type>java.lang.String</config-property-type>
    <config-property-value>"jdbc:mysql://mysql-servername:3306/databasename"</config-p
roperty-value>
    </config-property>
    ...
    </jndi-binding>
    ...
</jndi-bindings>
```

PostgreSQL

```
...
<jndi-bindings>
   <jndi-binding type="XAPooledDataSource"
    ...
    xa-datasource-class="org.postgresql.xa.PGXADataSource">
    <config-property>
    <config-property-name>ServerName</config-property-name>
    <config-property-type>java.lang.String</config-property-type>
    <config-property-value>postgresql-hostname</config-property-value>
    </config-property>
    <config-property>
    <config-property-name>DatabaseName</config-property-name>
    <config-property-type>java.lang.String</config-property-type>
    <config-property-value>postgresqldbname</config-property-value>
    </config-property>
    ...
   </jndi-binding>
    ...
</jndi-bindings>
```

Oracle

```
...
<jndi-bindings>
   <jndi-binding type="XAPooledDataSource"
    ...
    xa-datasource-class="oracle.jdbc.xa.client.OracleXADataSource">
    <config-property>
    <config-property-name>URL</config-property-name>
    <config-property-type>java.lang.String</config-property-type>
    <config-property-value>jdbc:oracle:oci8:@tc</config-property-value>
    </config-property>
    ...
    </jndi-binding>
    ...
</jndi-bindings>
```

Microsoft SQL Server

```
...
<jndi-bindings>
   <jndi-binding type="XAPooledDataSource"
      ...
    xa-datasource-class="com.microsoft.sqlserver.jdbc.SQLServerXADataSource">
    <config-property>
    <config-property-name>ServerName</config-property-name>
    <config-property-type>java.lang.String</config-property-type>
    <config-property-value>mysqlserver</config-property-value>
    </config-property>
    <config-property>
    <config-property-name>DatabaseName</config-property-name>
    <config-property-type>java.lang.String</config-property-type>
    <config-property-value>databasename</config-property-value>
    </config-property>
    <config-property>
    <config-property-name>SelectMethod</config-property-name>
    <config-property-type>java.lang.String</config-property-type>
    <config-property-value>cursor</config-property-value>
    </config-property>
    ...
    </jndi-binding>
    ...
</jndi-bindings>
```

ManagedDataSource Connection Example (Derby)

`ManagedDataSource` connections for the JCA `ManagedConnectionFactory` are configured as shown in the example. This configuration is similar to `XAPooledDataSource` connections, except the type is `ManagedDataSource`, and you specify a `managed-conn-factory-class` instead of an `xa-datasource-class`.

```
<?xml version="1.0"?>
<cache xmlns="http://geode.apache.org/schema/cache"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://geode.apache.org/schema/cache http://geode.apache.org/s
chema/cache/cache-1.0.xsd"
    version="1.0"
    lock-lease="120"
    lock-timeout="60"
    search-timeout="300">
   <region name="root">
      <region-attributes scope="distributed-no-ack" data-policy="cached" initial-capac
ity="16"
load-factor="0.75" concurrency-level="16" statistics-enabled="true">
      . . .
    </region>
    <jndi-bindings>
      <jndi-binding type="ManagedDataSource"
    jndi-name="DB3managed"
    init-pool-size="20"
    max-pool-size="100"
    idle-timeout-seconds="20"
    blocking-timeout-seconds="5"
    login-timeout-seconds="10"
    managed-conn-factory-class="com.myvendor.connection.ConnFactory"
    user-name="mitul"
    password="thecleartextpassword">
          <config-property>
             <config-property-name>Description</config-property-name>
             <config-property-type>java.lang.String</config-property-type>
             <config-property-value>pooled_transact</config-property-value>
          </config-property>
          <config-property>
             <config-property-name>DatabaseName</config-property-name>
             <config-property-type>java.lang.String</config-property-type>
             <config-property-value>newDB</config-property-value>
          </config-property>
          <config-property>
             <config-property-name>CreateDatabase</config-property-name>
             <config-property-type>java.lang.String</config-property-type>
             <config-property-value>create</config-property-value>
          </config-property>
           . . .
     </jndi-binding>
   </jndi-bindings>
 </cache>
```

PooledDataSource Example (Derby)

Use the `PooledDataSource` and `SimpleDataSource` connections for operations executed outside of any transaction. This example shows a `cache.xml` file configured for a pool of `PooledDataSource` connections to the data resource `newDB`. For this non-transactional connection pool, the log-in and blocking timeouts are set higher than for the transactional connection pools in the two previous examples. The connection information, including `user-name` and `password`, is set in the `cache.xml` file, instead of waiting until connection time. The password is not encrypted.

```
<?xml version="1.0"?>
<cache xmlns="http://geode.apache.org/schema/cache"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://geode.apache.org/schema/cache http://geode.apache.org/s
chema/cache/cache-1.0.xsd"
    version="1.0"
    lock-lease="120"
    lock-timeout="60"
    search-timeout="300">
    <region name="root">
         <region-attributes scope="distributed-no-ack" data-policy="cached"
initial-capacity="16" load-factor="0.75" concurrency-level="16" statistics-enabled="tr
ue">
            . . .
    </region>
    <jndi-bindings>
      <jndi-binding
    type="PooledDataSource"
    jndi-name="newDB1"
    init-pool-size="2"
    max-pool-size="7"
    idle-timeout-seconds="20"
    blocking-timeout-seconds="20"
    login-timeout-seconds="30"
    conn-pooled-datasource-class="org.apache.derby.jdbc.EmbeddedConnectionPoolDataSour
ce"
    user-name="mitul"
    password="thecleartextpassword">
       <config-property>
          <config-property-name>Description</config-property-name>
          <config-property-type>java.lang.String</config-property-type>
          <config-property-value>pooled_transact</config-property-value>
       </config-property>
       <config-property>
         <config-property-name>DatabaseName</config-property-name>
         <config-property-type>java.lang.String</config-property-type>
         <config-property-value>newDB</config-property-value>
       </config-property>
       <config-property>
         <config-property-name>CreateDatabase</config-property-name>
         <config-property-type>java.lang.String</config-property-type>
         <config-property-value>create</config-property-value>
       </config-property>
       . . .
      </jndi-binding>
   </jndi-bindings>
</cache>
```

SimpleDataSource Connection Example (Derby)

The example below shows a very basic configuration in the `cache.xml` file for a `SimpleDataSource` connection to the data resource `oldDB`. You only need to configure a few properties like a `jndi-name` for this connection pool, `oldDB1`, and the `databaseName`, `oldDB`. This password is in clear text.

A simple data source connection does not generally require vendor-specific property settings. If you need them, add `config-property` tags as shown in the earlier examples.

```
<?xml version="1.0"?>
<cache xmlns="http://geode.apache.org/schema/cache"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://geode.apache.org/schema/cache http://geode.apache.org/s
chema/cache/cache-1.0.xsd"
    version="1.0"
    lock-lease="120"
    lock-timeout="60"
    search-timeout="300">
   <region name="root">
      <region-attributes scope="distributed-no-ack" data-policy="cached" initial-capac
ity="16"
load-factor="0.75" concurrency-level="16" statistics-enabled="true">
        . . .
      </region-attributes>
    </region>
    <jndi-bindings>
      <jndi-binding type="SimpleDataSource"
    jndi-name="oldDB1"
    jdbc-driver-class="org.apache.derby.jdbc.EmbeddedDriver"
    user-name="mitul"
    password="thecleartextpassword"
    connection-url="jdbc:derby:newDB;create=true">
        . . .
       </jndi-binding>
   </jndi-bindings>
</cache>
```

How Data Loaders Work

By default, a region has no data loader defined. Plug an application-defined loader into any region by setting the region attribute cache-loader on the members that host data for the region.

The loader is called on cache misses during get operations, and it populates the cache with the new entry value in addition to returning the value to the calling thread.

A loader can be configured to load data into the VMware GemFire cache from an outside data store. To do the reverse operation, writing data from the VMware GemFire cache to an outside data store, use a cache writer event handler. See Implementing Cache Event Handlers.

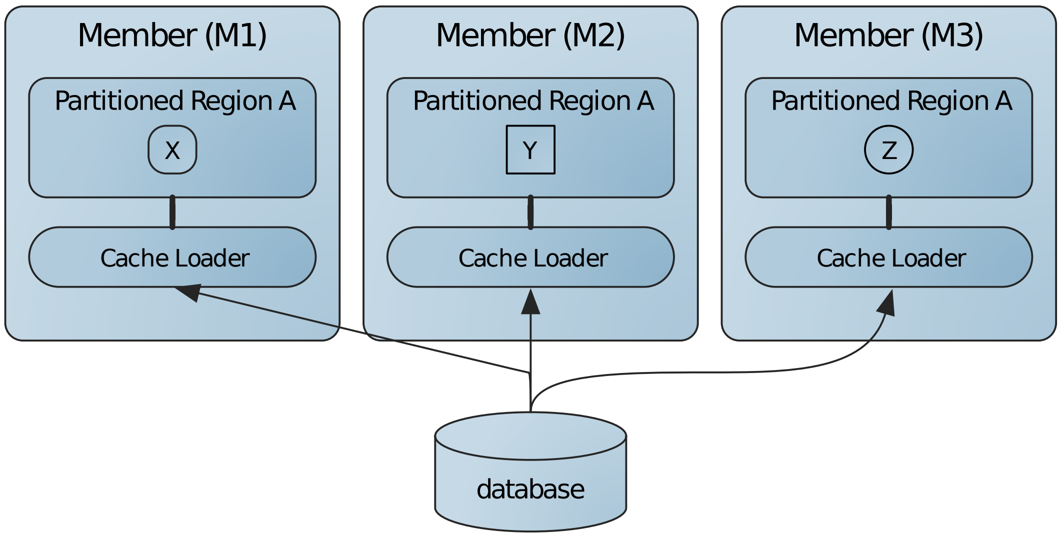
How to install your cache loader depends on the type of region.

Data Loading in Partitioned Regions

Because of the huge amounts of data they can handle, partitioned regions support partitioned loading. Each cache loader loads only the data entries in the member where the loader is defined. If data redundancy is configured, data is loaded only if the member holds the primary copy. So you must install a cache loader in every member where the partitioned attributes `local-max-memory` is not zero.

If you depend on a JDBC connection, every data store must have a connection to the data source, as shown in the following figure. Here the three members require three connections. See Configuring Database Connections Using JNDI for information on how to configure data sources.

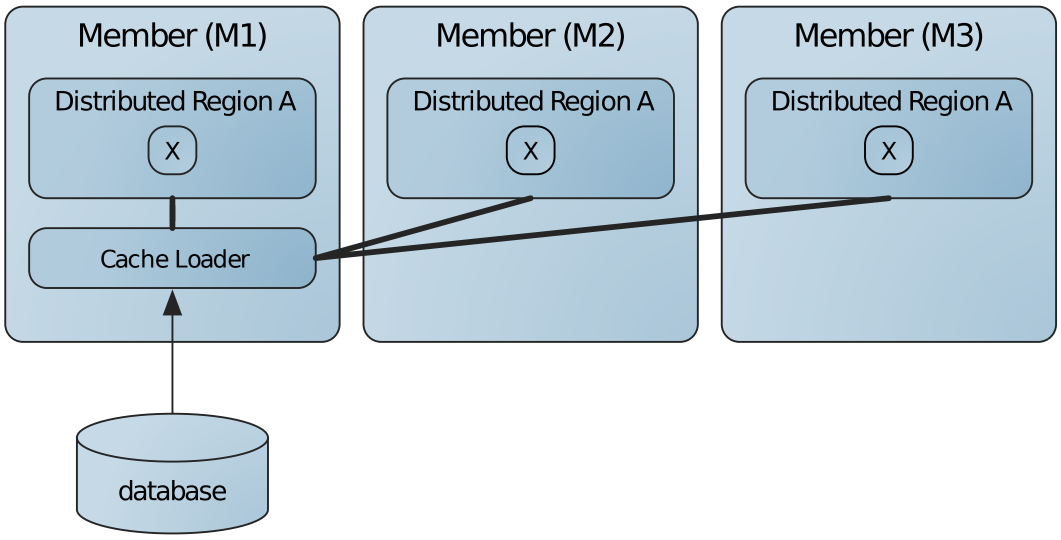
Note: Partitioned regions generally require more JDBC connections than distributed regions.



Data Loading in Distributed Regions

In a non-partitioned distributed region, a cache loader defined in one member is available to all members that have the region defined. Loaders are usually defined in just a subset of the caches holding the region. When a loader is needed, all available loaders for the region are invoked, starting with the most convenient loader, until the data is loaded or all loaders have been tried.

In the following figure, these members of one cluster can be running on different machines. Loading for the distributed region is performed from M1.



Data Loading in Local Regions

For local regions, the cache loader is available only in the member where it is defined. If a loader is defined, it is called whenever a value is not found in the local cache.

Implement a Data Loader

To use a data loader:

1. Implement the `org.apache.geode.cache.CacheLoader` interface.
2. Configure and deploy the implementation.

Implement the CacheLoader Interface

For a get operation, if the key is not in the cache, the thread serving the get operation invokes the `CacheLoader.load` method. Implement `load` to return the value for the key, which will be placed into the region in addition to being returned to the caller.

`org.apache.geode.cache.CacheLoader` inherits from `Declarable`, so implement the `Declarable.initialize` method if your `CacheLoader` implementation needs to be initialized with some arguments. Specify the required arguments either in your `cache.xml` file or in a gfsh `create region` or `alter region` command. Do not define the `Declarable.init()` method; it is deprecated.

Here is an example implementation:

```
public class SimpleCacheLoader implements CacheLoader {
    public Object load(LoaderHelper helper) {
        String key = (String) helper.getKey();

        // Return an entry value created from the key, assuming that
        // all keys are of the form "key1", "key2", "keyN"
        return "LoadedValue" + key.substring(3);
    }
}
```

If you need to run `Region` API calls from your implementation, spawn separate threads for them. Do not make direct calls to `Region` methods from your `load` method, as it could cause the cache loader to block, hurting the performance of the cluster.

Configure and Deploy

Use one of these three ways to configure and deploy the cache loader:

Option 1: If configuring a cluster by defining a `cache.xml` file, deploy by adding the cache loader to the classpath when starting servers.

Here is an example configuration within the `cache.xml` file that specifies the loader without arguments:

```
<region-attributes>
    <cache-loader>
        <class-name>myLoader</class-name>
    </cache-loader>
</region-attributes>
```

Or, here is an example configuration within the `cache.xml` file that specifies the loader with an argument:

```
<cache-loader>
    <class-name>com.company.data.DatabaseLoader</class-name>
    <parameter name="URL">
```

```
        <string>jdbc:cloudscape:rmi:MyData</string>
    </parameter>
</cache-loader>
```

To deploy the JAR file, add the cache loader JAR file to the classpath when starting servers. For example:

```
gfsh>start server --name=s2 --classpath=/var/data/lib/myLoader.jar
```

Option 2: If deploying the JAR file at server startup, add the JAR file to the classpath and use gfsh to apply the configuration to the region.

To deploy the JAR file, add the cache loader JAR file to the classpath when starting servers. For example:

```
gfsh>start server --name=s2 --classpath=/var/data/lib/myLoader.jar
```

Use gfsh to apply the configuration of the `CacheLoader` implementation to the region with `gfsh create region` or `gfsh alter region`. Here is an example of region creation without arguments:

```
gfsh>create region --name=r3 --cache-loader=com.example.appname.myCacheLoader
```

Here is an example of region creation with an argument:

```
gfsh>create region --name=r3 \
--cache-loader=com.example.appname.myCacheLoader{'URL':'jdbc:cloudscape:rmi:MyData'}
```

Here is an example of altering a region:

```
gfsh>alter region --name=r3 --cache-loader=com.example.appname.myCacheLoader
```

Option 3 applies to partitioned regions: If deploying the JAR file with the gfsh deploy command after servers have been started, use gfsh to apply the configuration to the region.

After server creation use gfsh to deploy the JAR file to all the servers. For example:

```
gfsh>deploy --jars=/var/data/lib/myLoader.jar
```

We do not generally use the gfsh deploy command when the servers host replicated regions, as detailed in How Data Loaders Work.

Use gfsh to apply the configuration of the `CacheLoader` implementation to the region with `gfsh create region` or `gfsh alter region`. Here is an example of region creation without arguments:

```
gfsh>create region --name=r3 --cache-loader=com.example.appname.myCacheLoader
```

Here is an example of region creation with an argument:

```
gfsh>create region --name=r3 \
--cache-loader=com.example.appname.myCacheLoader{'URL':'jdbc:cloudscape:rmi:MyData'}
```

Here is an example of altering a region:

```
gfsh>alter region --name=r3 --cache-loader=com.example.appname.myCacheLoader
```

Implementing a Server or Peer with a Cache Loader

Servers and peers with an embedded cache can configure a cache loader in only the members where it makes sense to do so. The design might, for example, assign the job of loading from a database to one or two members for a region hosted by many more members. This can be done to reduce the number of connections when the outside source is a database.

Implement the `org.apache.geode.cache.CacheLoader` interface. Region creation configures the the cache loader as in this example:

```
RegionFactory<String,Object> rf = cache.createRegionFactory(REPLICATE);
rf.setCacheLoader(new QuoteLoader());
quotes = rf.create("NASDAQ-Quotes");
```

Data Serialization

Data that you manage in VMware GemFire must be serialized and deserialized for storage and transmittal between processes. You can choose among several options for data serialization.

- **Overview of Data Serialization**

  VMware GemFire offers serialization options other than Java serialization that give you higher performance and greater flexibility for data storage, transfers, and language types.

- **VMware GemFire PDX Serialization**

  VMware GemFire's Portable Data eXchange (PDX) is a cross-language data format that can reduce the cost of distributing and serializing your objects. PDX stores data in named fields that you can access individually, to avoid the cost of deserializing the entire data object. PDX also allows you to mix versions of objects where you have added or removed fields.

- **VMware GemFire Data Serialization (DataSerializable and DataSerializer)**

  VMware GemFire's `DataSerializable` interface gives you quick serialization of your objects.

- **Standard Java Serialization**

  You can use standard Java serialization for data you only distribute between Java applications. If you distribute your data between non-Java clients and Java servers, you need to do additional programming to get the data between the various class formats.

Overview of Data Serialization

VMware GemFire offers serialization options other than Java serialization that give you higher performance and greater flexibility for data storage, transfers, and language types.

All data that VMware GemFire moves out of the local cache must be serializable. However, you do not necessarily need to implement `java.io.Serializable` since other serialization options are available in VMware GemFire. Region data that must be serializable falls under the following categories:

- Partitioned regions

- Distributed regions
- Regions that are persisted or overflowed to disk
- Server or client regions in a client/server installation
- Regions configured with a gateway sender for distributing events in a multi-site installation
- Regions that receive events from remote caches
- Regions that provide function arguments and results

Note: If you are storing objects with the HTTP Session Management Modules, these objects must be serializable since they are serialized before being stored in the region.

To minimize the cost of serialization and deserialization, VMware GemFire avoids changing the data format whenever possible. This means your data might be stored in the cache in serialized or deserialized form, depending on how you use it. For example, if a server acts only as a storage location for data distribution between clients, it makes sense to leave the data in serialized form, ready to be transmitted to clients that request it. Partitioned region data is always initially stored in serialized form.

Data Serialization Options

With VMware GemFire, you have the option to serialize your domain objects automatically or to implement serialization using one of VMware GemFire's interfaces. Enabling automatic serialization means that domain objects are serialized and deserialized without your having to make any code changes to those objects. This automatic serialization is performed by registering your domain objects with a custom `PdxSerializer` called the `ReflectionBasedAutoSerializer`, which uses Java reflection to infer which fields to serialize.

If autoserialization does not meet your needs, you can serialize your objects by implementing one of the VMware GemFire interfaces, `PdxSerializable` or `DataSerializable`. You can use these interfaces to replace any standard Java data serialization for better performance. If you cannot or do not want to modify your domain classes, each interface has an alternate serializer class, `PdxSerializer` and `DataSerializer`. To use these, you create your custom serializer class and then associate it with your domain class in the VMware GemFire cache configuration.

VMware GemFire Data serialization is about 25% faster than PDX serialization, however using PDX serialization will help you to avoid the even larger costs of performing deserialization.

** Serialization Options: Comparison of Features**

Capability	VMware GemFire Data Serializable	VMware GemFire PDX Serializable
Implements Java Serializable.	X	
Handles multiple versions of application domain objects, providing the versions differ by the addition or subtraction of fields.		X
Provides single field access of serialized data, without full deserialization - supported also for OQL querying.		X
Automatically ported to other languages by VMware GemFire		X
Works with .NET clients.	X	X

Capability	VMware GemFire Data Serializable	VMware GemFire PDX Serializable
Works with C++ clients.	X	X
Works with VMware GemFire delta propagation.	X	X (See note below.)

Note: By default, you can use VMware GemFire delta propagation with PDX serialization. However, delta propagation will not work if you have set the VMware GemFire property `read-serialized` to "true". In terms of deserialization, to apply a change delta propagation requires a domain class instance and the `fromDelta`method. If you have set `read-serialized` to true, then you will receive a `PdxInstance` instead of a domain class instance and `PdxInstance` does not have the `fromDelta` method required for delta propagation.

Differences between VMware GemFire Serialization (PDX or Data Serializable) and Java Serialization

VMware GemFire serialization (either PDX Serialization or Data Serialization) does not support circular object graphs whereas Java serialization does. In VMware GemFire serialization, if the same object is referenced more than once in an object graph, the object is serialized for each reference, and deserialization produces multiple copies of the object. By contrast in this situation, Java serialization serializes the object once and when deserializing the object, it produces one instance of the object with multiple references.

VMware GemFire PDX Serialization

VMware GemFire's Portable Data eXchange (PDX) is a cross-language data format that can reduce the cost of distributing and serializing your objects. PDX stores data in named fields that you can access individually, to avoid the cost of deserializing the entire data object. PDX also allows you to mix versions of objects where you have added or removed fields.

- **VMware GemFire PDX Serialization Features**

  VMware GemFire PDX serialization offers several advantages in terms of functionality.

- **High Level Steps for Using PDX Serialization**

  To use PDX serialization, you can configure and use VMware GemFire's reflection-based autoserializer, or you can program the serialization of your objects by using the PDX interfaces and classes.

- **Using Automatic Reflection-Based PDX Serialization**

  You can configure your cache to automatically serialize and deserialize domain objects without having to add any extra code to them.

- **Serializing Your Domain Object with a PdxSerializer**

  For a domain object that you cannot or do not want to modify, use the `PdxSerializer` class to serialize and deserialize the object's fields. You use one `PdxSerializer` implementation for the entire cache, programming it for all of the domain objects that you handle in this way.

- **Implementing PdxSerializable in Your Domain Object**

For a domain object with source that you can modify, implement the `PdxSerializable` interface in the object and use its methods to serialize and deserialize the object's fields.

- **Programming Your Application to Use PdxInstances**

  A `PdxInstance` is a light-weight wrapper around PDX serialized bytes. It provides applications with run-time access to fields of a PDX serialized object.

- **Adding JSON Documents to the VMware GemFire Cache**

  The `JSONFormatter` API allows you to put JSON formatted documents into regions and retrieve them later by storing the documents internally as PdxInstances.

- **Using PdxInstanceFactory to Create PdxInstances**

  You can use the `PdxInstanceFactory` interface to create a `PdxInstance` from raw data when the domain class is not available on the server.

- **Persisting PDX Metadata to Disk**

  VMware GemFire allows you to persist PDX metadata to disk and specify the disk store to use.

- **Using PDX Objects as Region Entry Keys**

  Using PDX objects as region entry keys is highly discouraged.

VMware GemFire PDX Serialization Features

VMware GemFire PDX serialization offers several advantages in terms of functionality.

Application Versioning of PDX Domain Objects

Domain objects evolve along with your application code. You might create an address object with two address lines, then realize later that a third line is required for some situations. Or you might realize that a particular field is not used and want to get rid of it. With PDX, you can use old and new versions of domain objects together in a cluster if the versions differ by the addition or removal of fields. This compatibility lets you gradually introduce modified code and data into the cluster, without bringing the cluster down.

VMware GemFire maintains a central registry of the PDX domain object metadata. Using the registry, VMware GemFire preserves fields in each member's cache regardless of whether the field is defined. When a member receives an object with a registered field that the member is not aware of, the member does not access the field, but preserves it and passes it along with the entire object to other members. When a member receives an object that is missing one or more fields according to the member's version, VMware GemFire assigns the Java default values for the field types to the missing fields.

Portability of PDX Serializable Objects

When you serialize an object using PDX, VMware GemFire stores the object's type information in the central registry. The information is passed among clients and servers, peers, and clusters.

This centralization of object type information is advantageous for client/server installations in which clients and servers are written in different languages. Clients pass registry information to servers automatically when they store a PDX serialized object. Clients can run queries and functions against the data in the servers without compatibility between server and the stored objects. One client can store data on the server to be retrieved by another client, with no requirements on the part of the server.

Reduced Deserialization of Serialized Objects

The access methods of PDX serialized objects allow you to examine specific fields of your domain object without deserializing the entire object. Depending on your object usage, you can reduce serialization and deserialization costs significantly.

Java and other clients can run queries and execute functions against the objects in the server caches without deserializing the entire object on the server side. The query engine automatically recognizes PDX objects, retrieves the `PdxInstance` of the object and uses only the fields it needs. Likewise, peers can access only the necessary fields from the serialized object, keeping the object stored in the cache in serialized form.

High Level Steps for Using PDX Serialization

To use PDX serialization, you can configure and use VMware GemFire's reflection-based autoserializer, or you can program the serialization of your objects by using the PDX interfaces and classes.

Optionally, program your application code to deserialize individual fields out of PDX representations of your serialized objects. You may also need to persist your PDX metadata to disk for recovery on startup.

Procedure

1. Use one of these serialization options for each object type that you want to serialize using PDX serialization:
    - Using Automatic Reflection-Based PDX Serialization
    - Serializing Your Domain Object with a PdxSerializer
    - Implementing PdxSerializable in Your Domain Object
2. To ensure that your servers do not need to load the application classes, set the `pdx read-serialized` attribute to true. In gfsh, execute the following command before starting up your servers:

    ```
    gfsh>configure pdx --read-serialized=true
    ```

    By using gfsh, this configuration can be propagated across the cluster through the Cluster Configuration Service. Alternately, you would need to configure `pdx read-serialized` in each server's `cache.xml` file.
3. If you are storing any VMware GemFire data on disk, then you must configure PDX serialization to use persistence. See Persisting PDX Metadata to Disk for more information.
4. (Optional) Wherever you run explicit application code to retrieve and manage your cached entries, you may want to manage your data objects without using full deserialization. To do this, see Programming Your Application to Use PdxInstances.

PDX and Multi-Site (WAN) Deployments

For multisite (WAN) installations only: If you will use PDX serialization in any of your WAN-enabled regions, for each cluster, you must choose a unique integer between 0 (zero) and 255 and set the `distributed-system-id` in every member's `gemfire.properties` file. See Configuring a Multi-site (WAN) System.

Using Automatic Reflection-Based PDX Serialization

You can configure your cache to automatically serialize and deserialize domain objects without having to add any extra code to them.

You can automatically serialize and deserialize domain objects without coding a `PdxSerializer` class. You do this by registering your domain objects with a custom `PdxSerializer` called `ReflectionBasedAutoSerializer` that uses Java reflection to infer which fields to serialize.

You can also extend the ReflectionBasedAutoSerializer to customize its behavior. For example, you could add optimized serialization support for BigInteger and BigDecimal types. See Extending the ReflectionBasedAutoSerializer for details.

Note: Your custom PDX autoserializable classes cannot use the `org.apache.geode` package. If they do, the classes will be ignored by the PDX auto serializer.

Prerequisites

- Understand generally how to configure the VMware GemFire cache.
- Understand how PDX serialization works and how to configure your application to use `PdxSerializer`.

Procedure

In your application where you manage data from the cache, provide the following configuration and code as appropriate:

1. In the domain classes that you wish to autoserialize, make sure each class has a zero-arg constructor. For example:

   ```
   public PortfolioPdx(){}
   ```

2. Using one of the following methods, set the PDX serializer to `ReflectionBasedAutoSerializer`.

   1. In gfsh, execute the following command prior to starting up any members that host data:

      ```
      gfsh>configure pdx --auto-serializable-classes=com\.company\.domain\..*
      ```

      By using gfsh, this configuration can propagated across the cluster through the Cluster Configuration Service.

   2. Alternately, in `cache.xml`:

      ```
      <!-- Cache configuration configuring auto serialization behavior -->
      <cache>
      ```

```
  <pdx>
    <pdx-serializer>
      <class-name>
       org.apache.geode.pdx.ReflectionBasedAutoSerializer
      </class-name>
      <parameter name="classes">
      <string>com.company.domain.DomainObject</string>
     </parameter>
  </pdx-serializer>
 </pdx>
  ...
</cache>
```

The parameter, `classes`, takes a comma-separated list of class patterns to define the domain classes to serialize. If your domain object is an aggregation of other domain classes, you need to register the domain object and each of those domain classes explicitly for the domain object to be serialized completely.

3. Using the Java API:

```
Cache c = new CacheFactory()
  .setPdxSerializer(new ReflectionBasedAutoSerializer("com.company.domai
n.DomainObject"))
  .create();
```

3. Customize the behavior of the `ReflectionBasedAutoSerializer` using one of the following mechanisms:
   - By using a class pattern string to specify the classes to auto-serialize and customize how the classes are serialized. Class pattern strings can be specified in the API by passing strings to the `ReflectionBasedAutoSerializer` constructor or by specifying them in cache.xml. See Customizing Serialization with Class Pattern Strings for details.
   - By creating a subclass of `ReflectionBasedAutoSerializer` and overriding specific methods. See Extending the ReflectionBasedAutoSerializer for details.
4. If desired, configure the `ReflectionBasedAutoSerializer` to check the portability of the objects it is passed before it tries to autoserialize them. When this flag is set to true, the `ReflectionBasedAutoSerializer` will throw a `NonPortableClassException` error when trying to autoserialize a non-portable object. To set this, use the following configuration:
   - In gfsh, use the following command:

```
gfsh>configure pdx --portable-auto-serializable-classes=com\.company\.dom
ain\..*
```

By using gfsh, this configuration can propagated across the cluster through the Cluster Configuration Service.

   - In cache.xml:

```
<!-- Cache configuration configuring auto serialization behavior -->
<cache>
  <pdx>
    <pdx-serializer>
```

```
      <class-name>
       org.apache.geode.pdx.ReflectionBasedAutoSerializer
      </class-name>
    <parameter name="classes">
      <string>com.company.domain.DomainObject</string>
    </parameter>
    <parameter name="check-portability">
      <string>true</string>
    </parameter>
  </pdx-serializer>
 </pdx>
  ...
</cache>
```

- Using the Java API:

```
Cache c = new CacheFactory()
  .setPdxSerializer(new ReflectionBasedAutoSerializer(true,"com.company.d
omain.DomainObject"))
  .create();
```

For each domain class you provide, all fields are considered for serialization except those defined as `static` or `transient` and those you explicitly exclude using the class pattern strings.

Note: The `ReflectionBasedAutoSerializer` traverses the given domain object's class hierarchy to retrieve all fields to be considered for serialization. So if `DomainObjectB` inherits from `DomainObjectA`, you only need to register `DomainObjectB` to have all of `DomainObjectB` serialized.

Customizing Serialization with Class Pattern Strings

Use class pattern strings to name the classes that you want to serialize using VMware GemFire's reflection-based autoserializer and to specify object identity fields and to specify fields to exclude from serialization.

The class pattern strings used to configured the `ReflectionBasedAutoSerializer` are standard regular expressions. For example, this expression would select all classes defined in the `com.company.domain` package and its subpackages:

```
com\.company\.domain\..*
```

You can augment the pattern strings with a special notation to define fields to exclude from serialization and to define fields to mark as PDX identity fields. The full syntax of the pattern string is:

```
<class pattern> [# (identity|exclude) = <field pattern>]... [, <class pattern>...]
```

The following example pattern string sets these PDX serialization criteria:

- Classes with names matching the pattern `com.company.DomainObject.*` are serialized. In those classes, fields beginning with `id` are marked as identity fields and fields named `creationDate` are not serialized.
- The class `com.company.special.Patient` is serialized. In the class, the field, `ssn` is marked as an identity field

```
com.company.DomainObject.*#identity=id.*#exclude=creationDate,
com.company.special.Patient#identity=ssn
```

Note: There is no association between the `identity` and `exclude` options, so the pattern above could also be expressed as:

```
com.company.DomainObject.*#identity=id.*, com.company.DomainObject.*#exclude=creationD
ate,
com.company.special.Patient#identity=ssn
```

Note: The order of the patterns is not relevant. All defined class patterns are used when determining whether a field should be considered as an identity field or should be excluded.

Examples:

- This XML uses the example pattern shown above:

  ```
  <parameter name="classes">
    <string>com.company.DomainObject.*#identity=id.*#exclude=creationDate,
  com.company.special.Patient#identity=ssn</string>
  </parameter>
  ```

- This application code sets the same pattern:

  ```
  classPatterns.add("com.company.DomainObject.*#identity=id.*#exclude=creationDat
  e,
     com.company.special.Patient#identity=ssn");
  ```

- This application code has the same effect:

  ```
   Cache c = new CacheFactory().set("cache-xml-file", cacheXmlFileName)
       .setPdxSerializer(new ReflectionBasedAutoSerializer("com.foo.DomainObject*
  #identity=id.*",
           "com.company.DomainObject.*#exclude=creationDate","com.company.specia
  l.Patient#identity=ssn"))
       .create();
  ```

Extending the ReflectionBasedAutoSerializer

You can extend the `ReflectionBasedAutoSerializer` to handle serialization in a customized manner. This section provides an overview of the available method-based customization options and an example of extending the serializer to support BigDecimal and BigInteger types.

Reasons to Extend the ReflectionBasedAutoSerializer

One of the main use cases for extending the `ReflectionBasedAutoSerializer` is that you want it to handle an object that would currently need to be handled by standard Java serialization. There are several issues with having to use standard Java serialization that can be addressed by extending the PDX `ReflectionBasedAutoSerializer`.

- Each time we transition from a VMware GemFire serialized object to an object that will be Java I/O serialized, extra data must get serialized. This can cause a great deal of serialization overhead. This is why it is worth extending the `ReflectionBasedAutoSerializer` to handle any classes that normally would have to be Java I/O serialized.

- Expanding the number of classes that can use the `ReflectionBasedAutoSerializer` is beneficial when you encounter object graphs. After we use Java I/O serialization on an object, any objects under that object in the object graph will also have to be Java I/O serialized. This includes objects that normally would have been serialized using PDX or `DataSerializable`.
- If standard Java I/O serialization is done on an object and you have enabled check-portability, then an exception will be thrown. Even if you are not concerned with the object's portability, you can use this flag to find out what classes would use standard Java serialization (by getting an exception on them) and then enhancing your auto serializer to handle them.

Overriding ReflectionBasedAutoSerializer Behavior

You can customize the specific behaviors in `ReflectionBasedAutoSerializer` by overriding the following methods:

- `isClassAutoSerialized` customizes which classes to autoserialize.
- `isFieldIncluded` specifies which fields of a class to autoserialize.
- `getFieldName` defines the specific field names that will be generated during autoserialization.
- `isIdentifyField` controls which field is marked as the identity field. Identity fields are used when a PdxInstance computes its hash code to determine whether it is equal to another object.
- `getFieldType` determines the field type that will be used when autoserializing the given field.
- `transformFieldValue` controls whether specific field values of a PDX object can be transformed during serialization.
- `writeTransform` controls what field value is written during auto serialization.
- `readTransform` controls what field value is read during auto deserialization.

These methods are only called the first time the `ReflectionBasedAutoSerializer` sees a new class. The results will be remembered and used the next time the same class is seen.

For details on these methods and their default behaviors, see the JavaDocs on ReflectionBasedAutoSerializer for details.

Example of Optimizing Autoserialization of BigInteger and BigDecimal Types

This section provides an example of extending the `ReflectionBasedAutoSerializer` to optimize the automatic serialization of BigInteger and BigDecimal types.

The following code sample illustrates a subclass of the `ReflectionBasedAutoSerializer` that optimizes BigInteger and BigDecimal autoserialization:

```
public static class BigAutoSerializer extends ReflectionBasedAutoSerializer {
   public BigAutoSerializer(Boolean checkPortability, string… patterns) {
    super(checkPortability, patterns);
}

@Override
public FieldType get FieldType(Field f, Class<?> clazz) {
   if (f.getType().equals(BigInteger.class)) {
        return FieldType.BYTE_ARRAY;
      } else if (f.getType().equals(BigDecimal.class)) {
        return FieldType.STRING;
      } else {
        return super.getFieldType(f, clazz);
      }
    }
@Override
    public boolean transformFieldValue(Field f, Class<?> clazz) {
      if (f.getType().equals(BigInteger.class)) {
        return true;
      } else if (f.getType().equals(BigDecimal.class)) {
        return true;
      } else {
        return super.transformFieldValue(f, clazz);
      }
    }

@Override
    public Object writeTransform(Field f, Class<?> clazz, Object originalValue) {
      if (f.getType().equals(BigInteger.class)) {
        byte[] result = null;
        if (originalValue != null) {
          BigInteger bi = (BigInteger)originalValue;
          result = bi.toByteArray();
        }
        return result;
      } else if (f.getType().equals(BigDecimal.class)) {
        Object result = null;
        if (originalValue != null) {
          BigDecimal bd = (BigDecimal)originalValue;
          result = bd.toString();
        }
        return result;
      } else {
        return super.writeTransform(f, clazz, originalValue);
      }
    }

@Override
    public Object readTransform(Field f, Class<?> clazz, Object serializedValue) {
      if (f.getType().equals(BigInteger.class)) {
        BigInteger result = null;
        if (serializedValue != null) {
          result = new BigInteger((byte[])serializedValue);
        }
        return result;
      } else if (f.getType().equals(BigDecimal.class)) {
        BigDecimal result = null;
        if (serializedValue != null) {
          result = new BigDecimal((String)serializedValue);
        }
        return result;
      } else {
        return super.readTransform(f, clazz, serializedValue);
      }
    }

  }
```

Serializing Your Domain Object with a PdxSerializer

For a domain object that you cannot or do not want to modify, use the `PdxSerializer` class to serialize and deserialize the object's fields. You use one `PdxSerializer` implementation for the entire cache, programming it for all of the domain objects that you handle in this way.

With `PdxSerializer`, you leave your domain object as-is and handle the serialization and deserialization in the separate serializer. You register the serializer in your cache PDX configuration. Program the serializer to handle all of the domain objects you need.

If you write your own `PdxSerializer` and you also use the `ReflectionBasedAutoSerializer`, then the `PdxSerializer` needs to own the `ReflectionBasedAutoSerializer` and delegate to it. A Cache can only have a single `PdxSerializer` instance.

Note: The `PdxSerializer` `toData` and `fromData` methods differ from those for `PdxSerializable`. They have different parameters and results.

Procedure

1. In the domain classes that you wish to PDX serialize, make sure each class has a zero-arg constructor. For example:

   ```
   public PortfolioPdx(){}
   ```

2. If you have not already implemented `PdxSerializer` for some other domain object, perform these steps:

   1. Create a new class as your cache-wide serializer and make it implement `PdxSerializer`. If you want to declare your new class in the `cache.xml` file, have it also implement `Declarable`.

      Example:

      ```
      import org.apache.geode.cache.Declarable;
      import org.apache.geode.pdx.PdxReader;
      import org.apache.geode.pdx.PdxSerializer;
      import org.apache.geode.pdx.PdxWriter;

      public class ExamplePdxSerializer implements PdxSerializer, Declarable {
      ...
      ```

   2. In your cache PDX configuration, register the serializer class in the cache's `<pdx>` `<pdx-serializer>` `<class-name>` attribute.

      Example:

```
// Configuration setting PDX serializer for the cache
<cache>
  <pdx>
    <pdx-serializer>
     <class-name>com.company.ExamplePdxSerializer</class-name>
    </pdx-serializer>
  </pdx>
  ...
</cache>
```

Or use the `CacheFactory.setPdxSerializer` API.

```
Cache c = new CacheFactory
   .setPdxSerializer(new ExamplePdxSerializer())
   .create();
```

Note: You cannot specify a custom `pdx-serializer` class using gfsh, however the `configure pdx` command automatically configures the org.apache.geode.pdx.ReflectionBasedAutoSerializer class. See configure pdx.

3. Program `PdxSerializer.toData` to recognize, cast, and handle your domain object:
    1. Write each standard Java data field of your domain class using the `PdxWriter` write methods.
    2. Call the `PdxWriter` `markIdentityField` method for each field you want to have VMware GemFire use to identify your object. Put this after the field's write method. VMware GemFire uses this information to compare objects for operations like distinct queries. If you do not set as least one identity field, then the `equals` and `hashCode` methods will use all PDX fields to compare objects and consequently, will not perform as well. It is important that the fields used by your `equals` and `hashCode` implementations are the same fields that you mark as identity fields.
    3. For a particular version of your class, you need to consistently write the same named field each time. The field names or number of fields must not change from one instance to another for the same class version.
    4. For best performance, do fixed width fields first and then variable length fields.
    5. If desired, you can check the portability of the object before serializing it by adding the `checkPortability` parameter when using the`PdxWriter` `writeObject`, `writeObjectArray`, and `writeField` methods.

    Example `toData` code:

```
public boolean toData(Object o, PdxWriter writer)
  {
    if(!(o instanceof PortfolioPdx)) {
      return false;
    }

    PortfolioPdx instance = (PortfolioPdx) o;
    writer.writeInt("id", instance.id)
    //identity field
    .markIdentityField("id")
    .writeDate("creationDate", instance.creationDate)
    .writeString("pkid", instance.pkid)
    .writeObject("positions", instance.positions)
    .writeString("type", instance.type)
    .writeString("status", instance.status)
    .writeStringArray("names", instance.names)
    .writeByteArray("newVal", instance.newVal)

    return true;
  }
```

1. Program `PdxSerializer.fromData` to create an instance of your class, read your data fields from the serialized form into the object's fields using the `PdxReader` read methods, and return the created object.

   Provide the same names that you did in `toData` and call the read operations in the same order as you called the write operations in your `toData` implementation.

   VMware GemFire provides the domain class type and `PdxReader` to the `fromData` method.

   Example `fromData` code:

   ```
     public Object fromData(Class<?> clazz, PdxReader reader)
     {
       if(!clazz.equals(PortfolioPdx.class)) {
         return null;
       }

       PortfolioPdx instance = new PortfolioPdx();
       instance.id = reader.readInt("id");
       instance.creationDate = reader.readDate("creationDate");
       instance.pkid = reader.readString("pkid");
       instance.positions = (Map<String, PositionPdx>)reader.readObject("pos
   itions");
       instance.type = reader.readString("type");
       instance.status = reader.readString("status");
       instance.names = reader.readStringArray("names");
       instance.newVal = reader.readByteArray("newVal");

       return instance;
     }
   ```

4. If desired, you can also enable extra validation in your use of `PdxWriter`. You can set this by setting the system property `gemfire.validatePdxWriters` to **true**. Note that you should only set this option if you are debugging new code as this option can decrease system performance.

Implementing PdxSerializable in Your Domain Object

For a domain object with source that you can modify, implement the `PdxSerializable` interface in the object and use its methods to serialize and deserialize the object's fields.

Procedure

1. In your domain class, implement `PdxSerializable`, importing the required `org.apache.geode.pdx` classes.

For example:

```
import org.apache.geode.pdx.PdxReader;
import org.apache.geode.pdx.PdxSerializable;
import org.apache.geode.pdx.PdxWriter;

public class PortfolioPdx implements PdxSerializable {
  ...
```

2. If your domain class does not have a zero-arg constructor, create one for it.

   For example:

   ```
   public PortfolioPdx(){}
   ```

3. Program `PdxSerializable.toData.`

   1. Write each standard Java data field of your domain class using the `PdxWriter` write methods. VMware GemFire automatically provides `PdxWriter` to the `toData` method for `PdxSerializable` objects.
   2. Call the `PdxWriter` `markIdentifyField` method for each field you want to have VMware GemFire use to identify your object. Put this after the field's write method. VMware GemFire uses this information to compare objects for operations like distinct queries. If you do not set as least one identity field, then the `equals` and `hashCode` methods will use all PDX fields to compare objects and consequently, will not perform as well. It is important that the fields used by your `equals` and `hashCode` implementations are the same fields that you mark as identity fields.
   3. For a particular version of your class, you need to consistently write the same named field each time. The field names or number of fields must not change from one instance to another for the same class version.
   4. For best performance, do fixed width fields first and then variable length fields.

      Example `toData` code:

      ```
      // PortfolioPdx fields
        private int id;
        private String pkid;
        private Map<String, PositionPdx> positions;
        private String type;
        private String status;
        private String[] names;
        private byte[] newVal;
        private Date creationDate;
        ...

        public void toData(PdxWriter writer)
        {
          writer.writeInt("id", id)
      // The markIdentifyField call for a field must
      // come after the field's write method
          .markIdentityField("id")
          .writeDate("creationDate", creationDate) //fixed length field
          .writeString("pkid", pkid)
          .writeObject("positions", positions)
      ```

```
      .writeString("type", type)
      .writeString("status", status)
      .writeStringArray("names", names)
      .writeByteArray("newVal", newVal)
    }
```

4. Program `PdxSerializable.fromData` to read your data fields from the serialized form into the object's fields using the `PdxReader` read methods.

   Provide the same names that you did in `toData` and call the read operations in the same order as you called the write operations in your `toData` implementation.

   VMware GemFire automatically provides `PdxReader` to the `fromData` method for `PdxSerializable` objects.

   Example `fromData` code:

```
public void fromData(PdxReader reader)
  {
    id = reader.readInt("id");
    creationDate = reader.readDate("creationDate");
    pkid = reader.readString("pkid");
    position1 = (PositionPdx)reader.readObject("position1");
    position2 = (PositionPdx)reader.readObject("position2");
    positions = (Map<String, PositionPdx>)reader.readObject("positions");
    type = reader.readString("type");
    status = reader.readString("status");
    names = reader.readStringArray("names");
    newVal = reader.readByteArray("newVal");
    arrayNull = reader.readByteArray("arrayNull");
    arrayZeroSize = reader.readByteArray("arrayZeroSize");
  }
```

What to do next

- As needed, configure and program your VMware GemFire applications to use `PdxInstance` for selective object deserialization. See Programming Your Application to Use PdxInstances.

Programming Your Application to Use PdxInstances

A `PdxInstance` is a light-weight wrapper around PDX serialized bytes. It provides applications with run-time access to fields of a PDX serialized object.

You can configure your cache to return a `PdxInstance` when a PDX serialized object is deserialized instead of deserializing the object to a domain class. You can then program your application code that reads your entries to handle `PdxInstance`s fetched from the cache.

Note: This applies only to entry retrieval that you explicitly code using methods like `EntryEvent.getNewValue` and `Region.get`, as you do inside functions or in cache listener code. This does not apply to querying because the query engine retrieves the entries and handles object access for you.

If you configure your cache to allow PDX serialized reads, a fetch from the cache returns the data in the form it is found. If the object is not serialized, the fetch returns the domain object. If the object is serialized, the fetch returns the `PdxInstance` for the object.

Note: If you are using `PdxInstance`s, you cannot use delta propagation to apply changes to PDX serialized objects.

For example, in client/server applications that are programmed and configured to handle all data activity from the client, PDX serialized reads done on the server side will always return a `PdxInstance`. This is because all of data is serialized for transfer from the client, and you are not performing any server-side activities that would deserialize the objects in the server cache.

In mixed situations, such as where a server cache is populated from client operations and also from data loads done on the server side, fetches done on the server can return a mix of `PdxInstance`s and domain objects.

When fetching data in a cache with PDX serialized reads enabled, the safest approach is to code to handle both types, receiving an `Object` from the fetch operation, checking the type and casting as appropriate. However, if you know that the class is not available in the JVM, then you can avoid performing the type check.

`PdxInstance` overrides any custom implementation you might have coded for your object's `equals` and `hashcode` methods. Make sure you have marked at least one identity field when writing PDX serialized objects. If you do not set as least one identity field, then the PdxInstance`equals` and `hashCode` methods will use all PDX fields to compare objects and consequently, will not perform as well.

Prerequisites

- Understand generally how to configure the VMware GemFire cache. See Basic Configuration and Programming.

Procedure

In your application where you fetch data from the cache, provide the following configuration and code as appropriate:

1. In the cache.xml file of the member where entry fetches are run, set the `<pdx> read-serialized` attribute to true. Data is not necessarily accessed on the member that you have coded for it. For example, if a client application runs a function on a server, the actual data access is done on the server, so you set `read-serialized` to true on the server.

   For example:

   ```
   // Cache configuration setting PDX read behavior
   <cache>
     <pdx read-serialized="true" />
     ...
   </cache>
   ```

2. Write the application code that fetches data from the cache to handle a `PdxInstance`. If you are sure you will only retrieve `PdxInstance`s from the cache, you can code only for that. In many cases, a `PdxInstance` or a domain object may be returned from your cache entry retrieval operation, so you should check the object type and handle each possible type.

   For example:

   ```
   // put/get code with serialized read behavior
   // put is done as normal
   ```

```
myRegion.put(myKey, myPdxSerializableObject);

// get checks Object type and handles each appropriately
Object myObject = myRegion.get(myKey);
if (myObject instanceof PdxInstance) {
  // get returned PdxInstance instead of domain object
  PdxInstance myPdxInstance = (PdxInstance)myObject;

  // PdxInstance.getField deserializes the field, but not the object
  String fieldValue = myPdxInstance.getField("stringFieldName");

  // Update a field and put it back into the cache
  // without deserializing the entire object
  WritablePdxInstance myWritablePdxI = myPdxInstance.createWriter();
  myWritablePdxI.setField("fieldName", fieldValue);
  region.put(key, myWritablePdxI);

  // Deserialize the entire object if needed, from the PdxInstance
  DomainClass myPdxObject = (DomainClass)myPdxInstance.getObject();
}
else if (myObject instanceof DomainClass) {
  // get returned instance of domain object
  // code to handle domain object instance
  ...
}
...
```

Note: Due to a limitation with PDX, if your PDX-enabled cache contains TreeSet domain objects, you should implement a Comparator that can handle both your domain objects and PdxInstance objects. You will also need to make the domain classes available on the server.

Adding JSON Documents to the Tanzu GemFire Cache

The `JSONFormatter` API allows you to put JSON formatted documents into regions and retrieve them later by storing the documents internally as PdxInstances.

VMware GemFire supports the use of JSON formatted documents natively. When you add a JSON document to a VMware GemFire cache, you call the JSONFormatter APIs to transform them into the PDX format (as a `PdxInstance`), which enables VMware GemFire to understand the JSON document at a field level.

In terms of querying and indexing, because the documents are stored internally as PDX, applications can index on any field contained inside the JSON document including any nested field (within JSON objects or JSON arrays.) Any queries run on these stored documents will return PdxInstances as results. To update a JSON document stored in VMware GemFire , you can execute a function on the PdxInstance.

You can then use the `JSONFormatter` to convert the PdxInstance results back into the JSON document.

`JSONFormatter` uses a streaming parser (Jackson, JSON processor) to turn JSON documents into the optimized PDX format. To use the JSONFormatter, make sure that `lib/geode-dependencies.jar` is available in your application's CLASSPATH.

The `JSONFormatter` class has four static methods that are used to convert JSON document into PdxInstances and then to convert those PdxInstances back into JSON document.

You need to call the following methods before putting any JSON document into the VMware GemFire region:

- `fromJSON`. Creates a PdxInstance from a JSON byte array. Returns the PdxInstance.
- `fromJSON`. Creates a PdxInstance from a JSON string. Returns the PdxInstance.

After putting the JSON document into a region as a PdxInstance, you can execute standard VMware GemFire queries and create indexes on the JSON document in the same manner you would query or index any other VMware GemFire PdxInstance.

After executing a VMware GemFire query or calling `region.get`, you can use the following methods to convert a PdxInstance back into the JSON format:

- `toJSON`. Reads a PdxInstance and returns a JSON string.
- `toJSONByteArray`. Reads a PdxInstance and returns a JSON byte array.

For more information on using the JSONFormatter, see the Java API documentation for `org.apache.geode.pdx.JSONFormatter`.

Sorting Behavior of Serialized JSON Fields

By default, VMware GemFire serialization creates a unique pdx typeID for each unique JSON document, even if the only difference between the JSON documents is the order in which their fields are specified.

If you prefer that JSON documents which differ only in the order in which their fields are specified map to the same typeID, set the property `gemfire.pdx.mapper.sort-json-field-names` to `true`. This tells the system to sort the JSON fields prior to serialization, allowing the system to identify matching entries, and helps reduce the number of pdx typeIDs that are generated by the serialization mechanism.

Using PdxInstanceFactory to Create PdxInstances

You can use the `PdxInstanceFactory` interface to create a `PdxInstance` from raw data when the domain class is not available on the server.

This can be particularly useful when you need an instance of a domain class for plug in code such as a function or a loader. If you have the raw data for the domain object (the class name and each field's type and data), then you can explicitly create a `PdxInstance`. The `PdxInstanceFactory` is very similar to the `PdxWriter` except that after writing each field, you need to call the create method which returns the created PdxInstance.

To create a factory call `RegionService.createPdxInstanceFactory`. A factory can only create a single instance. To create multiple instances create multiple factories or use `PdxInstance.createWriter()` to create subsequent instances. Using `PdxInstance.createWriter()` is usually faster.

When you create a PdxInstance, set as least one identity field using the `markIndentityField` method. If you do not mark an identity field, the PdxInstance`equals` and `hashCode` methods will use all PDX fields to compare objects and consequently, will not perform as well. It is important that the fields used by your `equals` and `hashCode` implementations are the same fields that you mark as identity fields.

The following is a code example of using `PdxInstanceFactory`:

```
PdxInstance pi = cache.createPdxInstanceFactory("com.company.DomainObject")
   .writeInt("id", 37)
   .markIdentityField("id")
   .writeString("name", "Mike Smith")
   .writeObject("favoriteDay", cache.createPdxEnum("com.company.Day", "FRIDAY", 5))
   .create();
```

For more information, see `PdxInstanceFactory` in the Java API documentation.

Enum Objects as PdxInstances

You can now work with enum objects as PdxInstances. When you fetch an enum object from the cache, you can now deserialize it as a `PdxInstance`. To check whether a `PdxInstance` is an enum, use the `PdxInstance.isEnum` method. An enum `PdxInstance` will have one field named “name” whose value is a String that corresponds to the enum constant name.

An enum `PdxInstance` is not writable; if you call `createWriter` it will throw an exception.

The `RegionService` has a method that allows you to create a `PdxInstance` that represents an enum. See `RegionService.createPdxEnum` in the Java API documentation.

Persisting PDX Metadata to Disk

VMware GemFire allows you to persist PDX metadata to disk and specify the disk store to use.

Prerequisites

- Understand generally how to configure the VMware GemFire cache. See Basic Configuration and Programming.
- Understand how VMware GemFire disk stores work. See Disk Storage.

Procedure

1. Set the `<pdx>` attribute `persistent` to true in your cache configuration. This is required for caches that use PDX with persistent regions and with regions that use a gateway sender to distribute events across a WAN.. Otherwise, it is optional.
2. (Optional) If you want to use a disk store that is not the VMware GemFire default disk store, set the `<pdx>` attribute `disk-store-name` to the name of your non-default disk store. **Note:** If you are using PDX serialized objects as region entry keys and you are using persistent regions, then you must configure your PDX disk store to be a different one than the disk store used by the persistent regions.
3. (Optional) If you later want to rename the PDX types that are persisted to disk, you can do so on your offline disk-stores by executing the `pdx rename` command. See pdx rename.

Example cache.xml:

This example `cache.xml` enables PDX persistence and sets a non-default disk store in a server cache configuration:

```
  <pdx read-serialized="true"
       persistent="true" disk-store-name="SerializationDiskStore">
```

```
    <pdx-serializer>
      <class-name>pdxSerialization.defaultSerializer</class-name>
    </pdx-serializer>
  </pdx>
  <region ...
```

Using PDX Objects as Region Entry Keys

Using PDX objects as region entry keys is highly discouraged.

The best practice for creating region entry keys is to use a simple key; for example, use a String or Integer. If the key must be a domain class, then you should use a non-PDX-serialized class.

If you must use PDX serialized objects as region entry keys, ensure that you do not set `read-serialized` to `true`. This configuration setting will cause problems in partitioned regions because partitioned regions require the hash code of the key to be the same on all JVMs in the distributed system. When the key is a `PdxInstance` object, its hash code will likely not be the same as the hash code of the domain object.

If you are using PDX serialized objects as region entry keys and you are using persistent regions, then you must configure your PDX disk store to be a different one than the disk store used by the persistent regions.

VMware GemFire Data Serialization (DataSerializable and DataSerializer)

VMware GemFire's `DataSerializable` interface gives you quick serialization of your objects.

Data Serialization with the DataSerializable Interface

VMware GemFire's `DataSerializable` interface gives you faster and more compact data serialization than the standard Java serialization or VMware GemFire PDX serialization. However, while VMware GemFire `DataSerializable` interface is generally more performant than VMware GemFire's `PdxSerializable`, it requires full deserialization on the server and then reserialization to send the data back to the client.

You can further speed serialization by registering the instantiator for your `DataSerializable` class through `Instantiator`, eliminating the need for reflection to find the right serializer. You can provide your own serialization through the API.

The recommended way to register your custom `Instantiator` is by specifying it in the `serialization-registration` element of cache.xml.

For more information, see the online Java documentation for `DataSerializable` and `DataSerializer`.

Example cache.xml:

The following provides an example of how to register an instantiator using cache.xml.

```
<serialization-registration>
<instantiator id="30">
   <class-name>com.package.MyClass</class-name>
```

```
</instantiator>
</serialization-registration>
```

In addition to speeding standard object serialization, you can use the `DataSerializable` interface to serialize any custom objects you store in the cache.

Serializing Your Domain Object with DataSerializer

You can also use `DataSerializer` to serialize domain objects. It serializes data in the same way as `DataSerializable` but allows you to serialize classes without modifying the domain class code.

See the JavaDocs on DataSerializable and DataSerializer for more information.

Standard Java Serialization

You can use standard Java serialization for data you only distribute between Java applications. If you distribute your data between non-Java clients and Java servers, you need to do additional programming to get the data between the various class formats.

Standard Java types are serializable by definition. For your domain classes, implement `java.io.Serializable`, then make sure to mark your transient and static variables as needed for your objects. For information, see the online documentation for `java.io.Serializable` for your Java version.

Mixing `DataSerializable` with `Serializable` or `PdxSerializable` use on the same data can result in increased memory use and lower throughput than using just `Serializable` on the entire data, especially if the `Serializable` entries are in collections. The bigger the data collection, the lower the throughput as the metadata for the collection entries is not shared when using `DataSerializable`.

Events and Event Handling

VMware GemFire provides versatile and reliable event distribution and handling for your cached data and system member events.

- **How Events Work**

  Members in your cluster receive cache updates from other members through cache events. The other members can be peers to the member, clients or servers or other clusters.

- **Implementing VMware GemFire Event Handlers**

  You can specify event handlers for region and region entry operations and for administrative events.

- **Configuring Peer-to-Peer Event Messaging**

  You can receive events from cluster peers for any region that is not a local region. Local regions receive only local cache events.

- **Configuring Client/Server Event Messaging**

  You can receive events from your servers for server-side cache events and query result changes.

- **Configuring Multi-Site (WAN) Event Queues**

  In a multi-site (WAN) installation, VMware GemFire uses gateway sender queues to distribute events for regions that are configured with a gateway sender. AsyncEventListeners also use an asynchronous event queue to distribute events for configured regions. This section describes additional options for configuring the event queues that are used by gateway senders or AsyncEventListener implementations.

How Events Work

Members in your VMware GemFire cluster receive cache updates from other members through cache events. The other members can be peers to the member, clients or servers or other clusters.

Events Features

These are the primary features of VMware GemFire events:

- Content-based events
- Asynchronous event notifications with conflation
- Synchronous event notifications for low latency
- High availability through redundant messaging queues
- Event ordering and once and only-once delivery
- Distributed event notifications
- Durable subscriptions
- Continuous querying

Types of Events

There are two categories of events and event handlers.

- Cache events in the caching API are used by applications with a cache. Cache events provide detail-level notification for changes to your data. Continuous query events are in this category.
- Administrative events in the administration API are used by administrative applications without caches.

Both kinds of events can be generated by a single member operation.

Note: You can handle one of these categories of events in a single system member. You cannot handle both cache and administrative events in a single member.

Because VMware GemFire maintains the order of administrative events and the order of cache events separately, using cache events and administrative events in a single process can cause unexpected results.

Event Cycle

The following steps describe the event cycle:

1. An operation begins, such as data put or a cache close.
2. The operation execution generates these objects:
    - An object of type `Operation` that describes the method that triggered the event.
    - An event object that describes the event, such as the member and region where the operation originated.
3. The event handlers that can handle the event are called and passed the event objects. Different event types require different handler types in different locations. If there is no matching event handler, that does not change the effect of the operation, which happens as usual.
4. When the handler receives the event, it triggers the handler's callback method for this event. The callback method can hand off the event object as input to another method. Depending on the type of event handler, the callbacks can be triggered before or after the operation. The timing depends on the event handler, not on the event itself. **Note:** For transactions, after-operation listeners receive the events after the transaction has committed.
5. If the operation is distributed, so that it causes follow-on operations in other members, those operations generate their own events, which can be handled by their listeners in the same way.

Event Objects

Event objects come in several types, depending on the operation. Some operations generate multiple objects of different types. All event objects contain data describing the event, and each event type carries slightly different kinds of data appropriate to its matching operation. An event object is stable. For example, its content does not change if you pass it off to a method on another thread.

For cache events, the event object describes the operation performed in the local cache. If the event originated remotely, it describes the local application of the remote entry operation, not the remote operation itself. The only exception is when the local region has an empty data policy; then the event carries the information for the remote (originating) cache operation.

Event Distribution

After a member processes an event in its local cache, it distributes it to remote caches according to the member's configuration and the configurations of the remote caches. For example, if a client updates its cache, the update is forwarded to the client's server. The server distributes the update to its peers and forwards it to any other clients according to their interest in the data entry. If the server system is part of a multi-site deployment and the data region is configured to use a gateway sender, then the gateway sender also forwards the update to a remote site, where the update is further distributed and propagated.

Event Handlers and Region Data Storage

You can configure a region for no local data storage and still send and receive events for the region. Conversely, if you store data in the region, the cache is updated with data from the event regardless of whether you have any event handlers installed.

Multiple Listeners

When multiple listeners are installed, as can be done with cache listeners, the listeners are invoked sequentially in the order they were added to the region or cache. Listeners are executed one at a time. So, unless you program a listener to pass off processing to another thread, you can use one listener's work in later listeners.

Event Ordering

During a cache operation, event handlers are called at various stages of the operation. Some event handlers are called before a region update and some are called after the region update operation. Depending on the type of event handler being called, the event handler can receive the events in-order or out-of-order in which they are applied on Region.

- `CacheWriter` and `AsyncEventListener` always receive events in the order in which they are applied on region.
- `CacheListener` and `CqListener` can receive events in a different order than the order in which they were applied on the region.

Note: An `EntryEvent` contains both the old value and the new value of the entry, which helps to indicate the value that was replaced by the cache operation on a particular key.

Peer-to-Peer Event Distribution

When a region or entry operation is performed, VMware GemFire distributes the associated events in the cluster according to system and cache configurations.

Install a cache listener for a region in each system member that needs to receive notification of region and entry changes.

Events in a Partitioned Region

A distributed operation follows this sequence in a partitioned region:

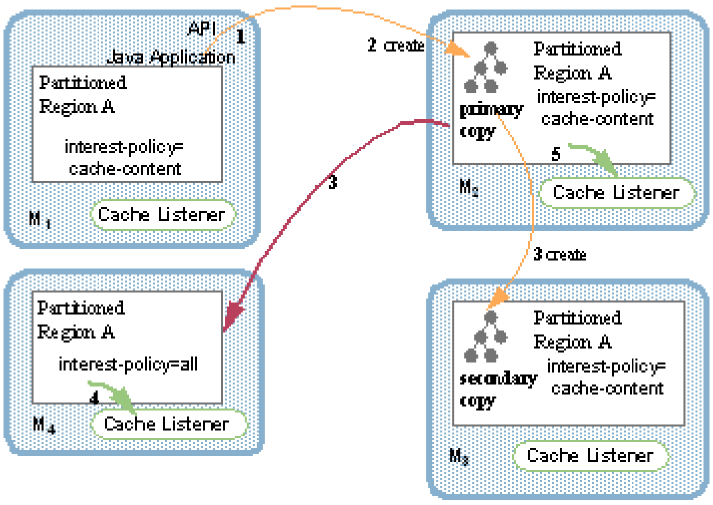
1. Apply the operation to the cache with the primary data entry, if appropriate.
2. Do the distribution based on the subscription-attributes interest-policy of the other members.
3. Invoke any listeners in the caches that receive the distribution.
4. Invoke the listener in the cache with the primary data entry.

In the following figure:

1. An API call in member M1 creates an entry.
2. The partitioned region creates the new entry in the cache in M2. M2, the holder of the primary copy, drives the rest of the procedure.
3. These two operations occur simultaneously:

- The partitioned region creates a secondary copy of the entry in the cache in M3. Creating the secondary copy does not invoke the listener on M3.
- M2 distributes the event to M4. This distribution to the other members is based on their interest policies. M4 has an interest-policy of all , so it receives notification of all events anywhere in the region. Since M1 and M3 have an interest-policy of cache-content , and this event does not affect any pre-existing entry in their local caches, they do not receive the event.

4. The cache listener on M4 handles the notification of the remote event on M2.
5. Once everything on the other members has completed successfully, the original create operation on M2 succeeds and invokes the cache listener on M2.



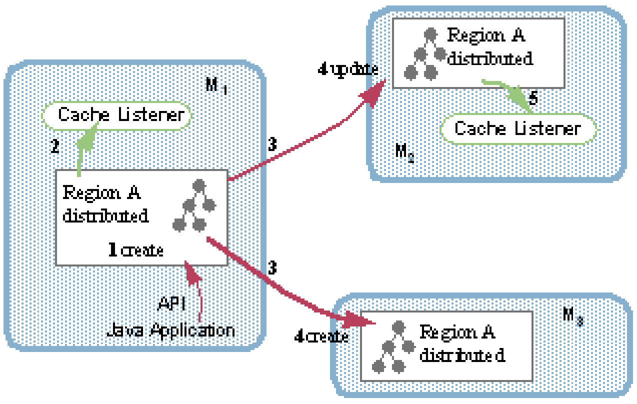
Events in a Distributed Region

A distributed operation follows this sequence in a distributed region:

1. Apply the operation to the local cache, if appropriate.
2. Invoke the local listeners.
3. Do the distribution.
4. Each member that receives the distribution carries out its own operation in response, which invokes any local listeners.

In the following figure:

1. An entry is created through a direct API call on member M1.
2. The create invokes the cache listener on M1.
3. M1 distributes the event to the other members.
4. M2 and M3 apply the remote change through their own local operations.
5. M3 does a create, but M2 does an update, because the entry already existed in its cache.
6. The cache listener on M2 receives callbacks for the local update. Since there is no cache listener on M3, the callbacks from the create on M3 are not handled. An API call in member M1 creates an entry.



Managing Events in Multi-threaded Applications

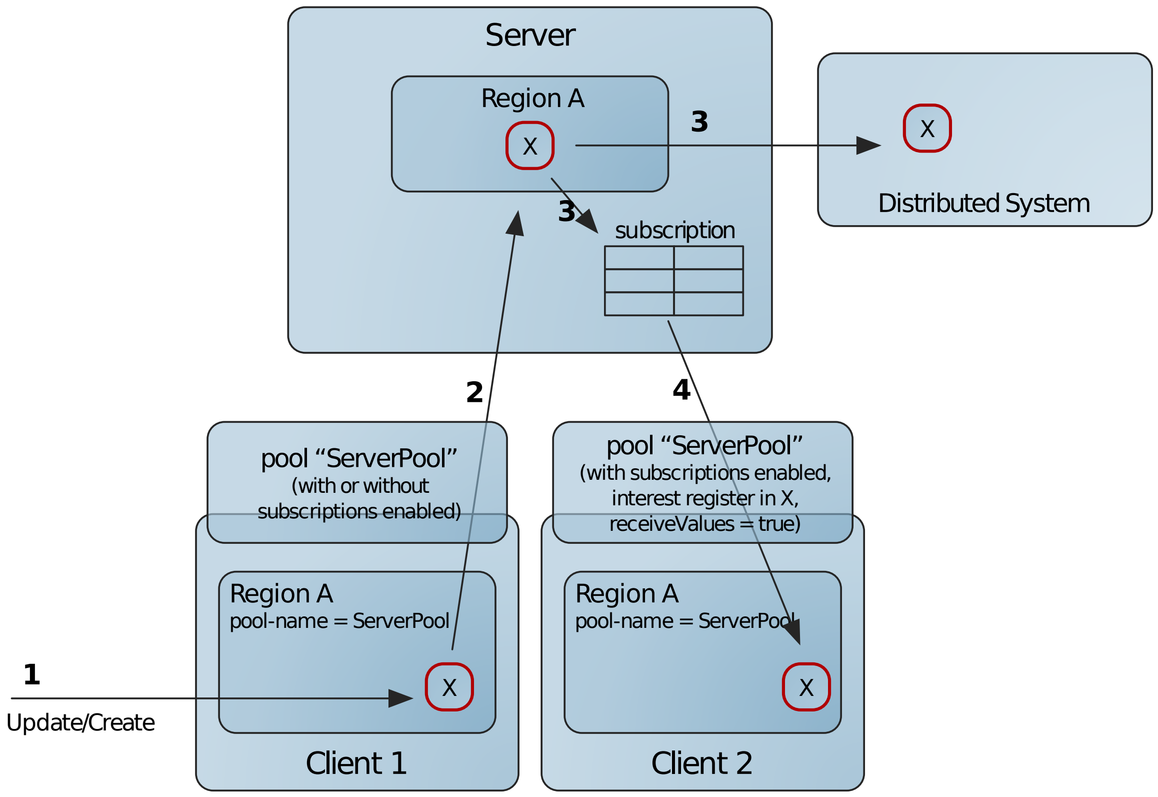
For partitioned regions, VMware GemFire guarantees ordering of events across threads, but for distributed regions it doesn't. For multi-threaded applications that create distributed regions, you need to use your application synchronization to make sure that one operation completes before the next one begins. Distribution through the distributed-no-ack queue can work with multiple threads if you set the `conserve-sockets` attribute to true. Then the threads share one queue, preserving the order of the events in distributed regions. Different threads can invoke the same listener, so if you allow different threads to send events, it can result in concurrent invocations of the listener. This is an issue only if the threads have some shared state - if they are incrementing a serial number, for example, or adding their events to a log queue. Then you need to make your code thread safe.

Client-to-Server Event Distribution

Clients and servers distribute events according to client activities and according to interest registered by the client in server-side cache changes.

When the client updates its cache, changes to client regions are automatically forwarded to the server side. The server-side update is then propagated to the other clients that are connected and have subscriptions enabled. The server does not return the update to the sending client.

The update is passed to the server and then passed, with the value, to every other client that has registered interest in the entry key. This figure shows how a client's entry updates are propagated.



The figure shows the following process:

1. Entry X is updated or created in Region A through a direct API call on Client1.
2. The update to the region is passed to the pool named in the region.
3. The pool propagates the event to the cache server, where the region is updated.
4. The server member distributes the event to its peers and also places it into the subscription queue for Client2 because that client has previously registered interest in entry X.
5. The event for entry X is sent out of the queue to Client2. When this happens is indeterminate.

Client to server distribution uses the client pool connections to send updates to the server. Any region with a named pool automatically forwards updates to the server. Client cache modifications pass first through a client `CacheWriter`, if one is defined, then to the server through the client pool, and then finally to the client cache itself. A cache writer, either on the client or server side, may cancel the operation.

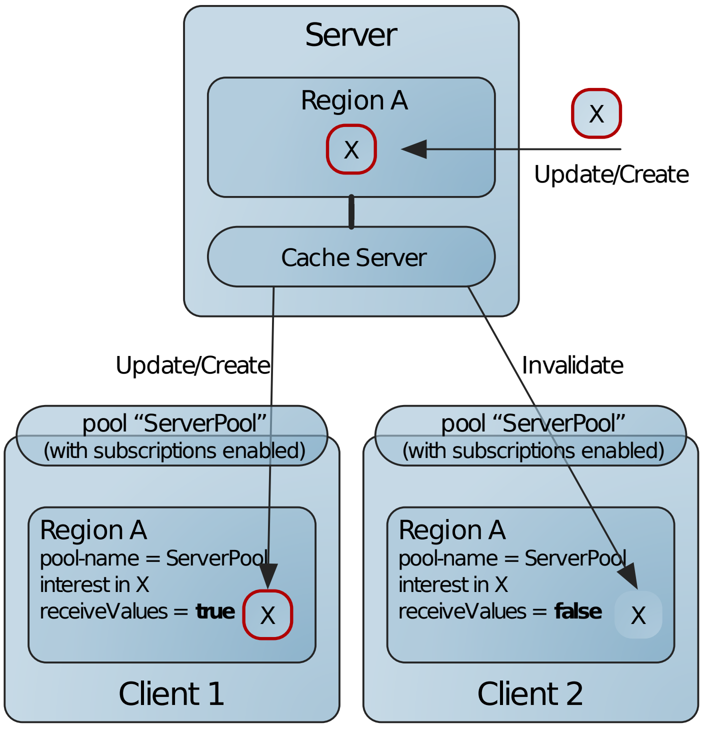
Change in Client Cache	Effect on Server Cache
Entry create or update	Creation or update of entry.
Distributed entry destroy	Entry destroy. The destroy call is propagated to the server even if the entry is not in the client cache.
Distributed region destroy/clear (distributed only)	Region destroy/clear

Note: Invalidations on the client side are not forwarded to the server.

Server-to-Client Event Distribution

The server automatically sends entry modification events only for keys in which the client has registered interest. In the interest registration, the client indicates whether to send new values or just invalidations for the server-side entry creates and updates. If invalidation is used, the client then updates the values lazily as needed.

This figure shows the complete event subscription event distribution for interest registrations, with value receipt requested (receiveValues=true) and without.



Change in Server Cache	Effect on Client Cache
Entry create/update	For subscriptions with `receiveValues` set to true, entry create or update. For subscriptions with `receiveValues` set to false, entry invalidate if the entry already exists in the client cache; otherwise, no effect. The next client get for the entry is forwarded to the server.
Entry invalidate/destroy (distributed only)	Entry invalidate/destroy
Region destroy/clear (distributed only)	Region destroy or local region clear

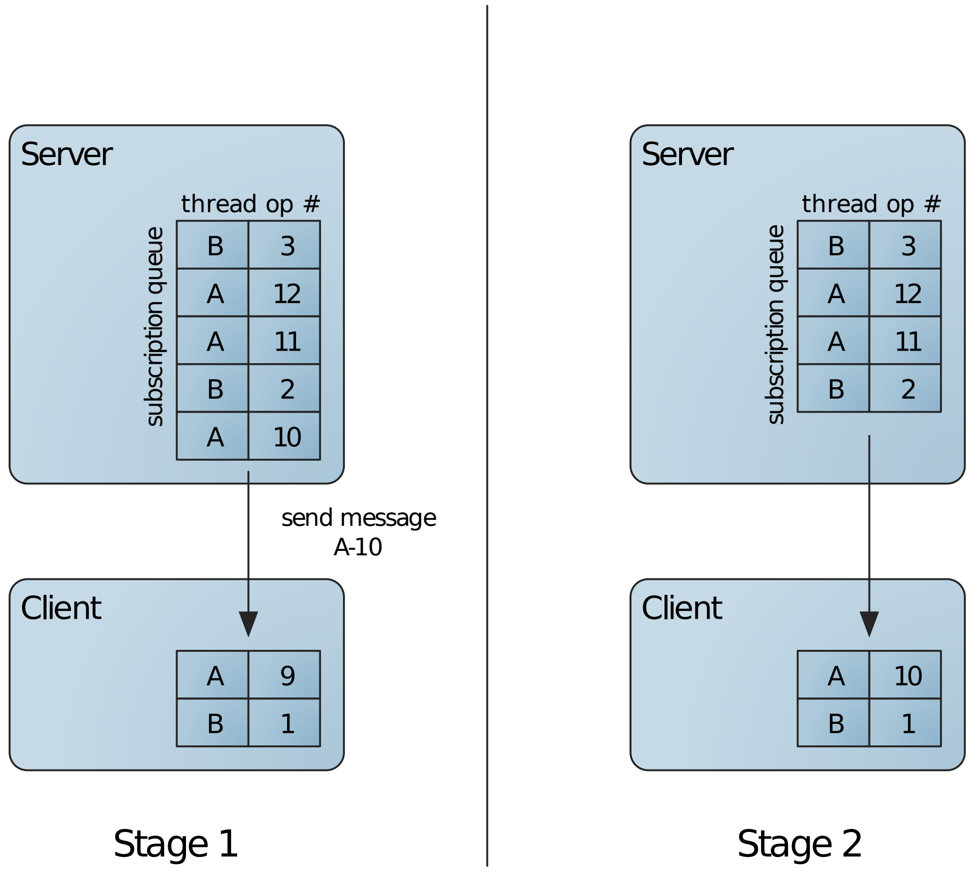
Server-side distributed operations are all operations that originate as a distributed operation in the server or one of its peers. Region invalidation in the server is not forwarded to the client.

Note: To maintain a unified set of data in your servers, do not do local entry invalidation in your server regions.

Server-to-Client Message Tracking

The server uses an asynchronous messaging queue to send events to its clients. Every event in the queue originates in an operation performed by a thread in a client, a server, or an application in the server's or some other cluster. The event message has a unique identifier composed of the originating thread's ID combined with its member's distributed system member ID, and the sequential ID of the operation. So the event messages originating in any single thread can be grouped and ordered by time from lowest sequence ID to highest. Servers and clients track the highest sequential ID for each member thread ID.

A single client thread receives and processes messages from the server, tracking received messages to make sure it does not process duplicate sends. It does this using the process IDs from originating threads.



The client's message tracking list holds the highest sequence ID of any message received for each originating thread. The list can become quite large in systems where there are many different threads coming and going and doing work on the cache. After a thread dies, its tracking entry is not needed. To avoid maintaining tracking information for threads that have died, the client expires entries that have had no activity for more than the `subscription-message-tracking-timeout`.

Client Interest Registration on the Server

The system processes client interest registration following these steps:

1. The entries in the client region that may be affected by this registration are silently destroyed. Other keys are left alone.

    - For the `registerInterest` method, the system destroys all of the specified keys, leaving other keys in the client region alone. So if you have a client region with keys A, B, and C and you register interest in the key list A, B, at the start of the `registerInterest` operation, the system destroys keys A and B in the client cache but does not touch key C.
    - For the `registerInterestRegex` method, the system silently destroys all keys in the client region.
2. The interest specification is sent to the server, where it is added to the client's interest list. The list can specify entries that are not in the server region at the time interest is registered.
3. If a bulk load is requested in the call's `InterestResultPolicy` parameter, before control is returned to the calling method, the server sends all data that currently satisfies the interest specification. The client's region is updated automatically with the downloaded data. If the server region is partitioned, the entire partitioned region is used in the bulk load. Otherwise, only the server's local cache region is used. The interest results policy options are:
    - KEYS—The client receives a bulk load of all available keys matching the interest registration criteria.
    - KEYS_VALUES—The client receives a bulk load of all available keys and values matching the interest registration criteria. This is the default interest result policy.
    - NONE—The client does not receive any immediate bulk loading.

Once interest is registered, the server continually monitors region activities and sends events to its clients that match the interest.

- No events are generated by the register interest calls, even if they load values into the client cache.
- The server maintains the union of all of the interest registrations, so if a client registers interest in key 'A', then registers interest in regular expression "B*", the server will send updates for all entries with key 'A' or key beginning with the letter 'B'.
- The server maintains the interest registration list separate from the region. The list can contain specifications for entries that are not currently in the server region.
- The `registerInterestRegex` method uses the standard `java.util.regex` methods to parse the key specification.

Server Failover

When a server hosting a subscription queue fails, the queueing responsibilities pass to another server. How this happens depends on whether the new server is a secondary server. In any case, all failover activities are carried out automatically by the VMware GemFire system.

- **Non-HA failover:** The client fails over without high availability if it is not configured for redundancy or if all secondaries also fail before new secondaries can be initialized. As soon as it can attach to a server, the client goes through an automatic reinitialization process. In this process, the failover code on the client side silently destroys all entries of interest to the client and refetches them from the new server, essentially reinitializing the client cache from the new server's cache. For the notify all configuration, this clears and reloads all of the entries for the client regions that are connected to the server. For notify by subscription, it clears and reloads only the entries in the region interest lists. To reduce failover noise, the events caused by the local entry destruction and refetching are blocked by the failover code and do not reach the client cache listeners. Because of this, your clients could receive some out-of-sequence events during and after a server failover. For example, entries that exist on the failed server and not on its replacement are destroyed and never recreated during a failover. Because the destruction events are blocked, the client ends up with entries removed from its cache with no associated destroy events.

- **HA failover:** If your client pool is configured with redundancy and a secondary server is available at the time the primary fails, the failover is invisible to the client. The secondary server resumes queueing activities as soon as the primary loss is detected. The secondary might resend a few events, which are discarded automatically by the client message tracking activities.

  Note: There is a very small potential for message loss during HA server failover. The risk is not present for failover to secondaries that have fully initialized their subscription queue data. The risk is extremely low in healthy systems that use at least two secondary servers. The risk is higher in unstable systems where servers often fail and where secondaries do not have time to initialize their subscription queue data before becoming primaries. To minimize the risk, the failover logic chooses the longest-lived secondary as the new primary.

  Note: Redundancy management is handled by the client, so when a durable client is disconnected from the server, client event redundancy is not maintained. Even if the servers fail one at a time, so that running clients have time to fail over and pick new secondary servers, an offline durable client cannot fail over. As a result, the client loses its queued messages.

Multi-Site (WAN) Event Distribution

VMware GemFire distributes a subset of cache events between clusters, with a minimum impact on each system's performance. Events are distributed only for regions that you configure to use a gateway sender for distribution.

Queuing Events for Distribution

In regions that are configured with one or more gateway senders (`gateway-sender-ids` attribute), events are automatically added to a gateway sender queue for distribution to other sites. Events that are placed in a gateway sender queue are distributed asynchronously to remote sites. For serial gateway queues, the ordering of events sent between sites can be preserved using the `order-policy` attribute.

If a queue becomes too full, it is overflowed to disk to keep the member from running out of memory. You can optionally configure the queue to be persisted to disk (with the `enable-persistence` `gateway-sender` attribute). With persistence, if the member that manages the queue goes down, the member picks up where it left off after it restarts.

Operation Distribution from a Gateway Sender

The multi-site installation is designed for minimal impact on cluster performance, so only the farthest-reaching entry operations are distributed between sites.

These operations are distributed:

- entry create
- entry put
- entry distributed destroy, providing the operation is not an expiration action

These operations are not distributed:

- get
- invalidate
- local destroy
- expiration actions of any kind
- region operations

How a Gateway Sender Processes Its Queue

Each primary gateway sender contains a processor thread that reads messages from the queue, batches them, and distributes the batches to a gateway receiver in a remote site. To process the queue, a gateway sender thread takes the following actions:

1. Reads messages from the queue
2. Creates a batch of the messages
3. Synchronously distributes the batch to the other site and waits for a reply
4. Removes the batch from the queue after the other site has successfully replied

Because the batch is not removed from the queue until after the other site has replied, the message cannot get lost. On the other hand, in this mode a message could be processed more than once. If a site goes offline in the middle of processing a batch of messages, then that same batch will be sent again once the site is back online.

You can configure the batch size for messages as well as the batch time interval settings. A gateway sender processes a batch of messages from the queue when either the batch size or the time interval is reached. In an active network, it is likely that the batch size will be reached before the time interval. In an idle network, the time interval will most likely be reached before the batch size. This may result in some network latency that corresponds to the time interval.

How a Gateway Sender Handles Batch Processing Failure

Exceptions can occur at different points during batch processing:

- The gateway receiver could fail with acknowledgment. If processing fails while the gateway receiver is processing a batch, the receiver replies with a failure acknowledgment that contains the exception, including the identity of the message that failed, and the ID of the last message that it successfully processed. The gateway sender then removes the successfully processed messages and the failed message from the queue and logs an exception with the failed message information. The sender then continues processing the messages remaining in the queue.

- The gateway receiver can fail without acknowledgment. If the gateway receiver does not acknowledge a sent batch, the gateway sender does not know which messages were successfully processed. In this case the gateway sender re-sends the entire batch.
- No gateway receivers may be available for processing. If a batch processing exception occurs because there are no remote gateway receivers available, then the batch remains in the queue. The gateway sender waits for a time, and then attempts to re-send the batch. The time period between attempts is five seconds. The existing server monitor continuously attempts to connect to the gateway receiver, so that a connection can be made and queue processing can continue. Messages build up in the queue and possibly overflow to disk while waiting for the connection.

List of Event Handlers and Events

VMware GemFire provides many types of events and event handlers to help you manage your different data and application needs.

Event Handlers

Use either cache handlers or membership handlers in any single application. Do not use both. The event handlers in this table are cache handlers unless otherwise noted.

Handler API	Events received	Description
`AsyncEventListener`	`AsyncEvent`	Tracks changes in a region for write-behind processing. Extends the `CacheCallback` interface. You install a write-back cache listener to an `AsyncEventQueue` instance. You can then add the `AsyncEventQueue` instance to one or more regions for write-behind processing. See Implementing an AsyncEventListener for Write-Behind Cache Event Handling.
`CacheCallback`		Superinterface of all cache event listeners. Functions only to clean up resources that the callback allocated.
`CacheListener`	`RegionEvent`, `EntryEvent`	Tracks changes to region and its data entries. Responds synchronously. Extends `CacheCallback` interface. Installed in region. Receives only local cache events. Install one in every member where you want the events handled by this listener. In a partitioned region, the cache listener only fires in the primary data store. Listeners on secondaries are not fired.
`CacheWriter`	`RegionEvent`, `EntryEvent`	Receives events for *pending* changes to the region and its data entries in this member or one of its peers. Has the ability to cancel the operations in question. Extends `CacheCallback` interface. Installed in region. Receives events from anywhere in the distributed region, so you can install one in one member for the entire distributed region. Receives events only in primary data store in partitioned regions, so install one in every data store.

Handler API	Events received	Description
`ClientMembershipListener`	`ClientMembershipEvent`	One of the interfaces that replaces the deprecated Admin APIs. You can use the ClientMembershipListener to receive membership events only about clients. This listener's callback methods are invoked when this process detects connection changes to clients. Callback methods include `memberCrashed`, `memberJoined`, `memberLeft` (graceful exit).
`CqListener`	`CqEvent`	Receives events from the server cache that satisfy a client-specified query. Extends `CacheCallback` interface. Installed in the client inside a `CqQuery`.
`GatewayConflictResolver`	`TimestampedEntryEvent`	Decides whether to apply a potentially conflicting event to a region that is distributed over a WAN configuration. This event handler is called only when the distributed system ID of an update event is different from the ID that last updated the region entry.
`MembershipListener`	`MembershipEvent`	Use this interface to receive membership events only about peers. This listener's callback methods are invoked when peer members join or leave the cluster. Callback methods include `memberCrashed`, `memberJoined`, and `memberLeft` (graceful exit).
`RegionMembershipListener`	`RegionEvent`	Provides after-event notification when a region with the same name has been created in another member and when other members hosting the region join or leave the cluster. Extends `CacheCallback` and `CacheListener`. Installed in region as a `CacheListener`.
`TransactionListener`	`TransactionEvent` with embedded list of `EntryEvent`	Tracks the outcome of transactions and changes to data entries in the transaction. **Note:** Multiple transactions on the same cache can cause concurrent invocation of `TransactionListener` methods, so implement methods that do the appropriate synchronizing of the multiple threads for thread-safe operation. Extends `CacheCallback` interface. Installed in cache using transaction manager. Works with region-level listeners if needed.
`TransactionWriter`	`TransactionEvent` with embedded list of `EntryEvent`	Receives events for *pending* transaction commits. Has the ability to cancel the transaction. Extends `CacheCallback` interface. Installed in cache using transaction manager. At most one writer is called per transaction. Install a writer in every transaction host.
`UniversalMembershipListenerAdapter`	`MembershipEvent` and `ClientMembershipEvent`	One of the interfaces that replaces the deprecated Admin APIs. Provides a wrapper for MembershipListener and ClientMembershipListener callbacks for both clients and peers.

Events

The events in this table are cache events unless otherwise noted.

Event	Passed to handler …	Description
`AsyncEvent`	`AsyncEventListener`	Provides information about a single event in the cache for asynchronous, write-behind processing.
`CacheEvent`		Superinterface to `RegionEvent` and `EntryEvent`. This defines common event methods, and contains data needed to diagnose the circumstances of the event, including a description of the operation being performed, information about where the event originated, and any callback argument passed to the method that generated this event.
`ClientMembershipEvent`	`ClientMembershipListener`	An event delivered to a `ClientMembershipListener` when this process detects connection changes to servers or clients.
`CqEvent`	`CqListener`	Provides information about a change to the results of a continuous query running on a server on behalf of a client. `CqEvent`s are processed on the client.
`EntryEvent`	`CacheListener`, `CacheWriter`, `TransactionListener` (inside the `TransactionEvent`)	Extends `CacheEvent` for entry events. Contains information about an event affecting a data entry in the cache. The information includes the key, the value before this event, and the value after this event. `EntryEvent.getNewValue` returns the current value of the data entry. `EntryEvent.getOldValue` returns the value before this event if it is available. For a partitioned region, returns the old value if the local cache holds the primary copy of the entry. `EntryEvent` provides the VMware GemFire transaction ID if available. You can retrieve serialized values from `EntryEvent` using the `getSerialized`* methods. This is useful if you get values from one region's events just to put them into a separate cache region. There is no counterpart `put` function as the put recognizes that the value is serialized and bypasses the serialization step.
`MembershipEvent` (membership event)	`MembershipListener`	An event that describes the member that originated this event. Instances of this are delivered to a `MembershipListener` when a member has joined or left the cluster.
`RegionEvent`	`CacheListener`, `CacheWriter`, `RegionMembershipListener`	Extends `CacheEvent` for region events. Provides information about operations that affect the whole region, such as reinitialization of the region after being destroyed.
`TimestampedEntryEvent`	`GatewayConflictResolver`	Extends `EntryEvent` to include a timestamp and distributed system ID associated with the event. The conflict resolver can compare the timestamp and ID in the event with the values stored in the entry to decide whether the local system should apply the potentially conflicting event.

Event	Passed to handler ...	Description
`TransactionEvent`	`TransactionListener`, `TransactionWriter`	Describes the work done in a transaction. This event may be for a pending or committed transaction, or for the work abandoned by an explicit rollback or failed commit. The work is represented by an ordered list of `EntryEvent` instances. The entry events are listed in the order in which the operations were performed in the transaction. As the transaction operations are performed, the entry events are conflated, with only the last event for each entry remaining in the list. So if entry A is modified, then entry B, then entry A, the list will contain the event for entry B followed by the second event for entry A.

Implementing VMware GemFire Event Handlers

You can specify event handlers for region and region entry operations and for administrative events.

- **Implementing Cache Event Handlers**

  Depending on your installation and configuration, cache events can come from local operations, peers, servers, and remote sites. Event handlers register their interest in one or more events and are notified when the events occur.

- **Implementing an AsyncEventListener for Write-Behind Cache Event Handling**

  An `AsyncEventListener` asynchronously processes batches of events after they have been applied to a region. You can use an `AsyncEventListener` implementation as a write-behind cache event handler to synchronize region updates with a database.

- **How to Safely Modify the Cache from an Event Handler Callback**

  Event handlers are synchronous. If you need to change the cache or perform any other distributed operation from event handler callbacks, be careful to avoid activities that might block and affect your overall system performance.

- **Cache Event Handler Examples**

  Some examples of cache event handlers.

Implementing Cache Event Handlers

Depending on your installation and configuration, cache events can come from local operations, peers, servers, and remote sites. Event handlers register their interest in one or more events and are notified when the events occur.

For each type of handler, VMware GemFire provides a convenience class with empty stubs for the interface callback methods.

Note: Write-behind cache listeners are created by extending the `AsyncEventListener` interface, and they are configured with an `AsyncEventQueue` that you assign to one or more regions.

Procedure

1. Decide which events your application needs to handle. For each region, decide which events you want to handle. For the cache, decide whether to handle transaction events.
2. For each event, decide which handlers to use. The `*Listener` and `*Adapter` classes in `org.apache.geode.cache.util` show the options.
3. Program each event handler:
    1. Extend the handler's adapter class.
    2. If you want to declare the handler in the `cache.xml`, implement the `org.apache.geode.cache.Declarable` interface as well.
    3. Implement the handler's callback methods as needed by your application.

       Note: Improperly programmed event handlers can block your distributed system. Cache events are synchronous. To modify your cache or perform distributed operations based on events, avoid blocking your system by following the guidelines in How to Safely Modify the Cache from an Event Handler Callback.

       Example:

       ```
       package myPackage;
       import org.apache.geode.cache.Declarable;
       import org.apache.geode.cache.EntryEvent;
       import org.apache.geode.cache.util.CacheListenerAdapter;
       import java.util.Properties;

       public class MyCacheListener extends CacheListenerAdapter implements Decl
       arable {
       /** Processes an afterCreate event.
        * @param event The afterCreate EntryEvent received
       */
         public void afterCreate(EntryEvent event) {
           String eKey = event.getKey();
           String eVal = event.getNewValue();
             ... do work with event info
         }
           ... process other event types
       }
       ```
4. Install the event handlers, either through the API or the `cache.xml`.

   XML Region Event Handler Installation:

   ```
   <region name="trades">
     <region-attributes ... >
       <!-- Cache listener -->
       <cache-listener>
         <class-name>myPackage.MyCacheListener</class-name>
       <cache-listener>
     </region-attributes>
   </region>
   ```

   Java Region Event Handler Installation:

```
tradesRegion = cache.createRegionFactory(RegionShortcut.PARTITION)
  .addCacheListener(new MyCacheListener())
  .create("trades");
```

XML Transaction Writer and Listener Installation:

```
<cache search-timeout="60">
      <cache-transaction-manager>
        <transaction-listener>
          <class-name>com.company.data.MyTransactionListener</class-name>
                <parameter name="URL">
                  <string>jdbc:cloudscape:rmi:MyData</string>
                </parameter>
           </transaction-listener>
           <transaction-listener>
            . . .
           </transaction-listener>
           <transaction-writer>
                <class-name>com.company.data.MyTransactionWriter</class-name>
                <parameter name="URL">
                    <string>jdbc:cloudscape:rmi:MyData</string>
                </parameter>
                <parameter
                  ...
                </parameter>
           </transaction-writer>
      </cache-transaction-manager>
      . . .
</cache>
```

The event handlers are initialized automatically during region creation when you start the member.

Installing Multiple Listeners on a Region

XML:

```
<region name="exampleRegion">
  <region-attributes>
    . . .
    <cache-listener>
      <class-name>myCacheListener1</class-name>
    </cache-listener>
    <cache-listener>
      <class-name>myCacheListener2</class-name>
    </cache-listener>
    <cache-listener>
      <class-name>myCacheListener3</class-name>
    </cache-listener>
  </region-attributes>
</region>
```

API:

```
CacheListener listener1 = new myCacheListener1();
CacheListener listener2 = new myCacheListener2();
CacheListener listener3 = new myCacheListener3();
```

```
Region nr = cache.createRegionFactory()
  .initCacheListeners(new CacheListener[]
    {listener1, listener2, listener3})
  .setScope(Scope.DISTRIBUTED_NO_ACK)
  .create(name);
```

Implementing an AsyncEventListener for Write-Behind Cache Event Handling

An `AsyncEventListener` asynchronously processes batches of events after they have been applied to a region. You can use an `AsyncEventListener` implementation as a write-behind cache event handler to synchronize region updates with a database.

How an AsyncEventListener Works

An `AsyncEventListener` instance is serviced by its own dedicated thread in which a callback method is invoked. Events that update a region are placed in an internal `AsyncEventQueue`, and one or more threads dispatch batches of events at a time to the listener implementation.

You can configure an `AsyncEventQueue` to be either serial or parallel. A serial queue is deployed to one VMware GemFire member, and it delivers all of a region's events, in order of occurrence, to a configured `AsyncEventListener` implementation. A parallel queue is deployed to multiple VMware GemFire members, and each instance of the queue delivers region events, possibly simultaneously, to a local `AsyncEventListener` implementation.

While a parallel queue provides the best throughput for writing events, it provides less control for ordering those events. With a parallel queue, you cannot preserve event ordering for a region as a whole because multiple VMware GemFire servers queue and deliver the region's events at the same time. However, the ordering of events for a given partition (or for a given queue of a distributed region) can be preserved.

For both serial and parallel queues, you can control the maximum amount of memory that each queue uses, as well as the batch size and frequency for processing batches in the queue. You can also configure queues to persist to disk (instead of simply overflowing to disk) so that write-behind caching can pick up where it left off when a member shuts down and is later restarted.

Optionally, a queue can use multiple threads to dispatch queued events. When you configure multiple threads for a serial queue, the logical queue that is hosted on a VMware GemFire member is divided into multiple physical queues, each with a dedicated dispatcher thread. You can then configure whether the threads dispatch queued events by key, by thread, or in the same order in which events were added to the queue. When you configure multiple threads for a parallel queue, each queue hosted on a VMware GemFire member is processed by dispatcher threads; the total number of queues created depends on the number of members that host the region.

A `GatewayEventFilter` can be placed on the `AsyncEventQueue` to control whether a particular event is sent to a selected `AsyncEventListener`. For example, events associated with sensitive data could be detected and not queued. For more detail, see the Javadocs for `GatewayEventFilter`.

A `GatewayEventSubstitutionFilter` can specify whether the event is transmitted in its entirety or in an altered representation. For example, to reduce the size of the data being serialized, it might be a more efficient to represent a full object by only its key. For more detail, see the Javadocs for `GatewayEventSubstitutionFilter`.

Operation Distribution from an AsyncEventQueue

An `AsyncEventQueue` distributes these operations:

- Entry create
- Entry put
- Entry distributed destroy, providing the operation is not an expiration action
- Expiration destroy, if the `forward-expiration-destroy` attribute is set to `true`. By default, this attribute is `false`, but you can set it to `true` using `cache.xml` or `gfsh`. To set this attribute in the Java API, use `AsyncEventQueueFactory.setForwardExpirationDestroy()`. See the javadocs for details.

These operations are not distributed:

- Get
- Invalidate
- Local destroy
- Region operations
- Expiration actions
- Expiration destroy, if the `forward-expiration-destroy` attribute is set to `false`. The default value is `false`.

Guidelines for Using an AsyncEventListener

Review the following guidelines before using an AsyncEventListener:

- If you use an `AsyncEventListener` to implement a write-behind cache listener, your code should check for the possibility that an existing database connection may have been closed due to an earlier exception. For example, check for `Connection.isClosed()` in a catch block and re-create the connection as needed before performing further operations.
- Use a serial `AsyncEventQueue` if you need to preserve the order of region events within a thread when delivering events to your listener implementation. Use parallel queues when the order of events within a thread is not important, and when you require maximum throughput for processing events. In both cases, serial and parallel, the order of operations on a given key is preserved within the scope of the thread.
- You must install the `AsyncEventListener` implementation on a VMware GemFire member that hosts the region whose events you want to process.
- If you configure a parallel `AsyncEventQueue`, deploy the queue on each VMware GemFire member that hosts the region.
- You can install a listener on more than one member to provide high availability and guarantee delivery for events, in the event that a member with the active `AsyncEventListener` shuts down. At any given time only one member has an active listener

for dispatching events. The listeners on other members remain on standby for redundancy. For best performance and most efficient use of memory, install only one standby listener (redundancy of at most one).

- Install no more than one standby listener (redundancy of at most one) for performance and memory reasons.
- To preserve pending events through member shutdowns, configure VMware GemFire to persist the internal queue of the `AsyncEventListener` to an available disk store. By default, any pending events that reside in the internal queue of an `AsyncEventListener` are lost if the active listener's member shuts down.
- To ensure high availability and reliable delivery of events, configure the event queue to be both persistent and redundant.

Implementing an AsyncEventListener

To receive region events for processing, you create a class that implements the `AsyncEventListener` interface. The `processEvents` method in your listener receives a list of queued `AsyncEvent` objects in each batch.

Each `AsyncEvent` object contains information about a region event, such as the name of the region where the event occurred, the type of region operation, and the affected key and value.

The basic framework for implementing a write-behind event handler involves iterating through the batch of events and writing each event to a database. For example:

```
class MyAsyncEventListener implements AsyncEventListener {

  public boolean processEvents(List<AsyncEvent> events) {

      // Process each AsyncEvent

      for(AsyncEvent event: events) {

          // Write the event to a database

      }
    }
}
```

Processing AsyncEvents

Use the AsyncEventListener.processEvents method to process AsyncEvents. This method is called asynchronously when events are queued to be processed. The size of the list reflects the number of batch events where batch size is defined in the AsyncEventQueueFactory. The `processEvents` method returns a boolean; true if the AsyncEvents are processed correctly, and false if any events fail processing. As long as `processEvents` returns false, VMware GemFire continues to re-try processing the events.

You can use the `getDeserializedValue` method to obtain cache values for entries that have been updated or created. Since the `getDeserializedValue` method will return a null value for destroyed entries, you should use the `getKey` method to obtain references to cache objects that have been destroyed. Here's an example of processing AsyncEvents:

```
public boolean processEvents(@SuppressWarnings("rawtypes") List<AsyncEvent> list)
 {
     logger.log (Level.INFO, String.format("Size of List<GatewayEvent> = %s", list.siz
e()));
     List<JdbcBatch> newEntries = new ArrayList<JdbcBatch>();

     List<JdbcBatch> updatedEntries = new ArrayList<JdbcBatch>();
     List<String> destroyedEntries = new ArrayList<String>();
     int possibleDuplicates = 0;

     for (@SuppressWarnings("rawtypes") AsyncEvent ge: list)
     {

       if (ge.getPossibleDuplicate())
        possibleDuplicates++;

       if ( ge.getOperation().equals(Operation.UPDATE))
       {
      updatedEntries.add((JdbcBatch) ge.getDeserializedValue());
       }
       else if ( ge.getOperation().equals(Operation.CREATE))
       {
         newEntries.add((JdbcBatch) ge.getDeserializedValue());
       }
       else if ( ge.getOperation().equals(Operation.DESTROY))
       {
      destroyedEntries.add(ge.getKey().toString());
       }

     }
```

Configuring an AsyncEventListener

To configure a write-behind cache listener, you first configure an asynchronous queue to dispatch the region events, and then create the queue with your listener implementation. You then assign the queue to a region in order to process that region's events.

Procedure

1. Configure a unique `AsyncEventQueue` with the name of your listener implementation. You can optionally configure the queue for parallel operation, persistence, batch size, and maximum memory size. See WAN Configuration for more information.

   gfsh configuration

   ```
   gfsh>create async-event-queue --id=sampleQueue --persistent --disk-store=exampl
   eStore --listener=com.myCompany.MyAsyncEventListener --listener-param=url#jdbc:
   db2:SAMPLE,username#gfeadmin,password#admin1
   ```

   The parameters for this command uses the following syntax:

   ```
   create async-event-queue --id=value --listener=value [--group=value] [--batch-s
   ize=value]
   ```

```
[--persistent(=value)?] [--disk-store=value] [--max-queue-memory=value] [--list
ener-param=value(,value)*]
```

For more information, see create async-event-queue.

cache.xml Configuration

```
<cache>
   <async-event-queue id="sampleQueue" persistent="true"
    disk-store-name="exampleStore" parallel="false">
      <async-event-listener>
         <class-name>MyAsyncEventListener</class-name>
         <parameter name="url">
           <string>jdbc:db2:SAMPLE</string>
         </parameter>
         <parameter name="username">
           <string>gfeadmin</string>
         </parameter>
         <parameter name="password">
           <string>admin1</string>
         </parameter>
      </async-event-listener>
    </async-event-queue>
...
</cache>
```

Java Configuration

```
Cache cache = new CacheFactory().create();
AsyncEventQueueFactory factory = cache.createAsyncEventQueueFactory();
factory.setPersistent(true);
factory.setDiskStoreName("exampleStore");
factory.setParallel(false);
AsyncEventListener listener = new MyAsyncEventListener();
AsyncEventQueue asyncQueue = factory.create("sampleQueue", listener);
```

2. If you are using a parallel `AsyncEventQueue`, the gfsh example above requires no alteration, as gfsh applies to all members. If using cache.xml or the Java API to configure your `AsyncEventQueue`, repeat the above configuration in each VMware GemFire member that will host the region. Use the same ID and configuration settings for each queue configuration. **Note:** You can ensure other members use the sample configuration by using the cluster configuration service available in gfsh. See Overview of the Cluster Configuration Service.

3. On each VMware GemFire member that hosts the `AsyncEventQueue`, assign the queue to each region that you want to use with the `AsyncEventListener` implementation.

   gfsh Configuration

   ```
   gfsh>create region --name=Customer --async-event-queue-id=sampleQueue
   ```

   Note that you can specify multiple queues on the command line in a comma-delimited list.

   cache.xml Configuration

   ```
   <cache>
   <region name="Customer">
   ```

```
    <region-attributes async-event-queue-ids="sampleQueue">
    </region-attributes>
  </region>
...
</cache>
```

Java Configuration

```
RegionFactory rf1 = cache.createRegionFactory();
rf1.addAsyncEventQueue(sampleQueue);
Region customer = rf1.create("Customer");

// Assign the queue to multiple regions as needed
RegionFactory rf2 = cache.createRegionFactory();
rf2.addAsyncEventQueue(sampleQueue);
Region order = rf2.create("Order");
```

Using the Java API, you can also add and remove queues to regions that have already been created:

```
AttributesMutator mutator = order.getAttributesMutator();
mutator.addAsyncEventQueueId("sampleQueue");
```

See the VMware GemFire API documentation for more information.

4. Optionally configure persistence and conflation for the queue. **Note:** You must configure your AsyncEventQueue to be persistent if you are using persistent data regions. Using a non-persistent queue with a persistent region is not supported.
5. Optionally configure multiple dispatcher threads and the ordering policy for the queue using the instructions in Configuring Dispatcher Threads and Order Policy for Event Distribution.

The `AsyncEventListener` receives events from every region configured with the associated `AsyncEventQueue`.

How to Safely Modify the Cache from an Event Handler Callback

Event handlers are synchronous. If you need to change the cache or perform any other distributed operation from event handler callbacks, be careful to avoid activities that might block and affect your overall system performance.

Operations to Avoid in Event Handlers

Do not perform distributed operations of any kind directly from your event handler. VMware GemFire is a highly distributed system and many operations that may seem local invoke distributed operations.

These are common distributed operations that can get you into trouble:

- Calling `Region` methods, on the event's region or any other region.
- Using the VMware GemFire `DistributedLockService`.
- Modifying region attributes.

- Executing a function through the VMware GemFire `FunctionService`.

To be on the safe side, do not make any calls to the VMware GemFire API directly from your event handler. Make all VMware GemFire API calls from within a separate thread or executor.

How to Perform Distributed Operations Based on Events

If you need to use the VMware GemFire API from your handlers, make your work asynchronous to the event handler. You can spawn a separate thread or use a solution like the `java.util.concurrent.Executor` interface.

This example shows a serial executor where the callback creates a `Runnable` that can be pulled off a queue and run by another object. This preserves the ordering of events.

```
public void afterCreate(EntryEvent event) {
  final Region otherRegion = cache.getRegion("/otherRegion");
  final Object key = event.getKey();
  final Object val = event.getNewValue();

  serialExecutor.execute(new Runnable() {
    public void run() {
      try {
        otherRegion.create(key, val);
      }
      catch (org.apache.geode.cache.RegionDestroyedException e) {
        ...
      }
      catch (org.apache.geode.cache.EntryExistsException e) {
        ...
      }
    }
  });
  }
```

For additional information on the `Executor`, see the `SerialExecutor` example on the Oracle Java web site.

Cache Event Handler Examples

Some examples of cache event handlers.

Declaring and Loading an Event Handler with Parameters

This declares an event handler for a region in the `cache.xml`. The handler is a cache listener designed to communicate changes to a DB2 database. The declaration includes the listener's parameters, which are the database path, username, and password.

```
<region name="exampleRegion">
  <region-attributes>
  . . .
    <cache-listener>
      <class-name>JDBCListener</class-name>
      <parameter name="url">
        <string>jdbc:db2:SAMPLE</string>
      </parameter>
      <parameter name="username">
        <string>gfeadmin</string>
      </parameter>
      <parameter name="password">
        <string>admin1</string>
      </parameter>
    </cache-listener>
  </region-attributes>
  </region>
```

This code listing shows part of the implementation of the `JDBCListener` declared in the `cache.xml`. This listener implements the `Declarable` interface. When an entry is created in the cache, this listener's `afterCreate` callback method is triggered to update the database. Here the listener's properties, provided in the `cache.xml`, are passed into the `Declarable.init` method and used to create a database connection.

```
. . .
public class JDBCListener
extends CacheListenerAdapter
implements Declarable {
  public void afterCreate(EntryEvent e) {
  . . .
    // Initialize the database driver and connection using input parameters
    Driver driver = (Driver) Class.forName(DRIVER_NAME).newInstance();
    Connection connection =
      DriverManager.getConnection(_url, _username, _password);
      System.out.println(_connection);
        . . .
  }
    . . .
  public void init(Properties props) {
    this._url = props.getProperty("url");
    this._username = props.getProperty("username");
    this._password = props.getProperty("password");
  }
}
```

Installing an Event Handler Through the API

This listing defines a cache listener using the `RegionFactory` method `addCacheListener`.

```
Region newReg = cache.createRegionFactory()
          .addCacheListener(new SimpleCacheListener())
          .create(name);
```

You can create a cache writer similarly, using the `RegionFactory` method `setCacheWriter`, like this:

```
Region newReg = cache.createRegionFactory()
          .setCacheWriter(new SimpleCacheWriter())
          .create(name);
```

Installing Multiple Listeners on a Region

XML:

```
<region name="exampleRegion">
  <region-attributes>
    . . .
    <cache-listener>
      <class-name>myCacheListener1</class-name>
    </cache-listener>
    <cache-listener>
      <class-name>myCacheListener2</class-name>
    </cache-listener>
    <cache-listener>
      <class-name>myCacheListener3</class-name>
    </cache-listener>
  </region-attributes>
</region>
```

API:

```
CacheListener listener1 = new myCacheListener1();
CacheListener listener2 = new myCacheListener2();
CacheListener listener3 = new myCacheListener3();

Region nr = cache.createRegionFactory()
  .initCacheListeners(new CacheListener[]
    {listener1, listener2, listener3})
  .setScope(Scope.DISTRIBUTED_NO_ACK)
  .create(name);
```

Installing a Write-Behind Cache Listener

```
//AsyncEventQueue with listener that performs WBCL work
<cache>
   <async-event-queue id="sampleQueue" persistent="true"
    disk-store-name="exampleStore" parallel="false">
      <async-event-listener>
         <class-name>MyAsyncListener</class-name>
         <parameter name="url">
           <string>jdbc:db2:SAMPLE</string>
         </parameter>
         <parameter name="username">
           <string>gfeadmin</string>
         </parameter>
         <parameter name="password">
           <string>admin1</string>
         </parameter>
               </async-event-listener>
             </async-event-queue>

// Add the AsyncEventQueue to region(s) that use the WBCL
  <region name="data">
       <region-attributes async-event-queue-ids="sampleQueue">
    </region-attributes>
  </region>
</cache>
```

Configuring Peer-to-Peer Event Messaging

You can receive events from cluster peers for any region that is not a local region. Local regions receive only local cache events.

Peer distribution is done according to the region's configuration.

- Replicated regions always receive all events from peers and require no further configuration. Replicated regions are configured using the `REPLICATE` region shortcut settings.
- For non-replicated regions, decide whether you want to receive all entry events from the distributed cache or only events for the data you have stored locally. To configure:
    - To receive all events, set the `subscription-attributes` `interest-policy` to `all`:

      ```
      <region-attributes>
          <subscription-attributes interest-policy="all"/>
      </region-attributes>
      ```

    - To receive events just for the data you have stored locally, set the `subscription-attributes` `interest-policy` to `cache-content` or do not set it (`cache-content` is the default):

      ```
      <region-attributes>
          <subscription-attributes interest-policy="cache-content"/>
      </region-attributes>
      ```

  For partitioned regions, this only affects the receipt of events, as the data is stored according to the region partitioning. Partitioned regions with interest policy of `all` can create network bottlenecks, so if you can, run listeners in every member that hosts the partitioned region data and use the `cache-content` interest policy.

Note: You can also configure Regions using the gfsh command-line interface. See Region Commands.

Configuring Client/Server Event Messaging

You can receive events from your servers for server-side cache events and query result changes.

For cache updates, you can configure to receive entry keys and values or just entry keys, with the data retrieved lazily when requested. The queries are run continuously against server cache events, with the server sending the deltas for your query result sets.

Before you begin, set up your client/server installation and configure and program your basic event messaging.

Servers receive updates for all entry events in their client's client regions.

To receive entry events in the client from the server:

1. Set the client pool `subscription-enabled` to true. See <pool>.

2. Program the client to register interest in the entries you need.

   Note: This must be done through the API.

   Register interest in all keys, a key list, individual keys, or by comparing key strings to regular expressions. By default, no entries are registered to receive updates. Specify whether the server is to send values with entry update events. Interest registration is only available through the API.

   1. Get an instance of the region where you want to register interest.
   2. Use the region's `registerInterest`* methods to specify the entries you want. Examples:

      ```
      // Register interest in a single key and download its entry
      // at this time, if it is available in the server cache
      Region region1 = . . . ;
      region1.registerInterest("key-1");

      // Register Interest in a List of Keys but do not do an initial bulk load
      // do not send values for creater/update events - just send key with inva
      lidation
      Region region2 = . . . ;
      List list = new ArrayList();
      list.add("key-1");
      list.add("key-2");
      list.add("key-3");
      list.add("key-4");
      region2.registerInterestForKeys(list, InterestResultPolicy.NONE, false);

      // Register interest in all keys and download all available keys now
      Region region3 = . . . ;
      region3.registerInterestForAllKeys(InterestResultPolicy.KEYS);

      // Register Interest in all keys matching a regular expression
      Region region1 = . . . ;
      region1.registerInterestRegex("[a-zA-Z]+_[0-9]+");
      ```

      You can call the register interest methods multiple times for a single region. Each interest registration adds to the server's list of registered interest criteria for the client. So if a client registers interest in key 'A', then registers interest in regular expression "B*", the server will send updates for all entries with key 'A' or key beginning with the letter 'B'.

   3. For highly available event messaging, configure server redundancy. See Configuring Highly Available Servers.
   4. To have events enqueued for your clients during client downtime, configure durable client/server messaging.
   5. Write any continuous queries (CQs) that you want to run to receive continuously streaming updates to client queries. CQ events do not update the client cache. If you have dependencies between CQs and/or interest registrations, so that you want the two types of subscription events to arrive as closely together on the client, use a single server pool for everything. Using different pools can lead to time

differences in the delivery of events because the pools might use different servers to process and deliver the event messages.

Configuring Highly Available Servers

With highly-available servers, one of the backups steps in and takes over messaging with no interruption in service if the client's primary server crashes.

To configure high availability, set the `subscription-redundancy` in the client's pool configuration. This setting indicates the number of secondary servers to use. For example:

```
<!-- Run one secondary server -->
<pool name="red1" subscription-enabled="true" subscription-redundancy="1">
  <locator host="nick" port="41111"/>
  <locator host="nora" port="41111"/>
</pool>
```

```
<!-- Use all available servers as secondaries. One is primary, the rest are secondarie
s -->
<pool name="redX" subscription-enabled="true" subscription-redundancy="-1">
  <locator host="nick" port="41111"/>
  <locator host="nora" port="41111"/>
</pool>
```

When redundancy is enabled, secondary servers maintain queue backups while the primary server pushes events to the client. If the primary server fails, one of the secondary servers steps in as primary to provide uninterrupted event messaging to the client.

The following table describes the different values for the subscription-redundancy setting:

subscription-redundancy	Description
0	No secondary servers are configured, so high availability is disabled.
> 0	Sets the precise number of secondary servers to use for backup to the primary.
-1	Every server that is not the primary is to be used as a secondary.

Highly Available Client/Server Event Messaging

With server redundancy, each pool has a primary server and some number of secondaries. The primaries and secondaries are assigned on a per-pool basis and are generally spread out for load balancing, so a single client with multiple pools may have primary queues in more than one server.

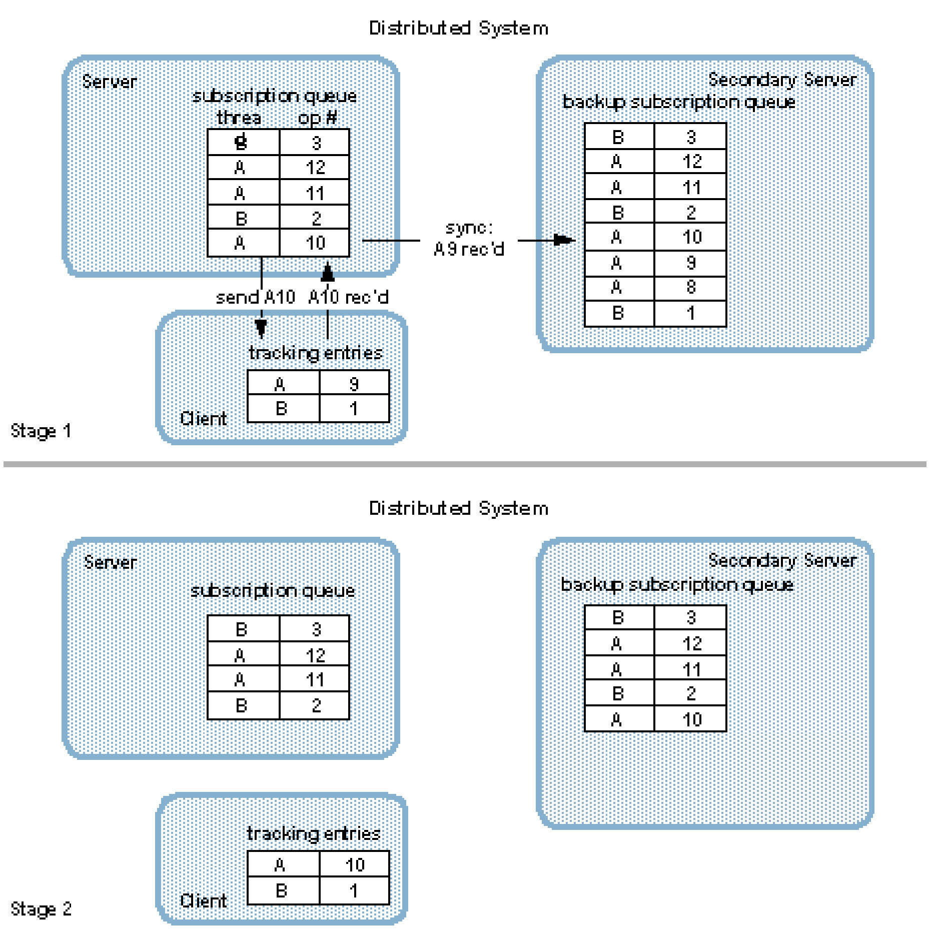
The primary server pushes events to clients and the secondaries maintain queue backups. If the primary server fails, one of the secondaries becomes primary to provide uninterrupted event messaging.

For example, if there are six servers running and `subscription-redundancy` is set to two, one server is the primary, two servers are secondary, and the remaining three do not actively participate in HA for the client. If the primary server fails, the system assigns one of the secondaries as the new primary and attempts to add another server to the secondary pool to retain the initial redundancy level. If no new secondary server is found, then the redundancy level is not satisfied but the failover procedure completes successfully. As soon as another secondary is available, it is added.

When high availability is enabled:

- The primary server sends event messages to the clients.
- Periodically, the clients send received messages to the server and the server removes the sent messages from its queues.
- Periodically, the primary server synchronizes with its secondaries, notifying them of messages that can be discarded because they have already been sent and received. There is a lag in notification, so the secondary servers remain only roughly synchronized with the primary. Secondary queues contain all messages that are contained in the primary queue plus possibly a few messages that have already been sent to clients.
- In the case of primary server failure, one of the secondaries becomes the primary and begins sending event messages from its queues to the clients. Immediately after failover, the new primary usually resends some messages that were already sent by the old primary. The client recognizes these as duplicates and discards them.

In stage 1 of this figure, the primary sends an event message to the client and a synchronization message to its secondary. By stage 2, the secondary and client have updated their queue and message tracking information. If the primary failed at stage two, the secondary would start sending event messages from its queue beginning with message A10. The client would discard the resend of message A10 and then process subsequent messages as usual.



Change Server Queue Synchronization Frequency

By default, the primary server sends queue synchronization messages to the secondaries every second. You can change this interval with the `gfsh alter runtime` command

Set the interval for queue synchronization messages as follows:

- gfsh:

  ```
  gfsh>alter runtime --message-sync-interval=2
  ```

- XML:

  ```
  <!-- Set sync interval to 2 seconds -->
  <cache ... message-sync-interval="2" />
  ```

- Java:

  ```
  cache = CacheFactory.create();
  cache.setMessageSyncInterval(2);
  ```

The ideal setting for this interval depends in large part on your application behavior. These are the benefits of shorter and longer interval settings:

- A shorter interval requires less memory in the secondary servers because it reduces queue buildup between synchronizations. In addition, fewer old messages in the secondary queues means reduced message re-sends after a failover. These considerations are most important for systems with high data update rates.
- A longer interval requires fewer distribution messages between the primary and secondary, which benefits overall system performance.

Set Frequency of Orphan Removal from the Secondary Queues

Usually, all event messages are removed from secondary subscription queues based on the primary's synchronization messages. Occasionally, however, some messages are orphaned in the secondary queues. For example, if a primary fails in the middle of sending a synchronization message to its secondaries, some secondaries might receive the message and some might not. If the failover goes to a secondary that did receive the message, the system will have secondary queues holding messages that are no longer in the primary queue. The new primary will never synchronize on these messages, leaving them orphaned in the secondary queues.

To make sure these messages are eventually removed, the secondaries expire all messages that have been enqueued longer than the time indicated by the servers' `message-time-to-live`.

Set the time-to-live as follows:

- XML:

  ```
  <!-- Set message ttl to 5 minutes -->
  <cache-server port="41414" message-time-to-live="300" />
  ```

- Java:

```
Cache cache = ...;
CacheServer cacheServer = cache.addCacheServer();
cacheServer.setPort(41414);
cacheServer.setMessageTimeToLive(200);
cacheServer.start();
```

Implementing Durable Client/Server Messaging

Use durable messaging for subscriptions that you need maintained for your clients even when your clients are down or disconnected. You can configure any of your event subscriptions as durable. Events for durable queries and subscriptions are saved in a queue when the client is disconnected and played back when the client reconnects. Other queries and subscriptions are removed from the queue.

Use durable messaging for client/server installations that use event subscriptions.

These are the high-level tasks described in this topic:

1. Configure your client as durable
2. Decide which subscriptions should be durable and configure accordingly
3. Program your client to manage durable messaging for disconnect, reconnect, and event handling

Configure the Client as Durable

Use one of the following methods:

- `gemfire.properties` file:

  ```
  durable-client-id=31
  durable-client-timeout=200
  ```

- Java:

  ```
  Properties props = new Properties();
  props.setProperty("durable-client-id", "31");
  props.setProperty("durable-client-timeout", "" + 200);
  CacheFactory cf = new CacheFactory(props);
  ```

The `durable-client-id` indicates that the client is durable and gives the server an identifier to correlate the client to its durable messages. For a non-durable client, this id is an empty string. The ID can be any number that is unique among the clients attached to servers in the same cluster.

The `durable-client-timeout` tells the server how long to wait for client reconnect. When this timeout is reached, the server stops storing to the client's message queue and discards any stored messages. The default is 300 seconds. This is a tuning parameter. If you change it, take into account the normal activity of your application, the average size of your messages, and the level of risk you can handle, both in lost messages and in the servers' capacity to store enqueued messages. Assuming that no messages are being removed from the queue, how long can the server run before the queue reaches the maximum capacity? How many durable clients can the server handle? To assist with tuning, use the VMware GemFire message queue statistics for durable clients through the disconnect and reconnect cycles.

Configure Durable Subscriptions and Continuous Queries

The register interest and query creation methods all have an optional boolean parameter for indicating durability. By default all are non-durable.

```
// Durable registration
// Define keySpecification, interestResultPolicy, durability
exampleRegion.registerInterest(keySpecification, interestResultPolicySpecification, tr
ue);

// Durable CQ
// Define cqName, queryString, cqAttributes, durability
CqQuery myCq = queryService.newCq(cqName, queryString, cqAttributes, true);
```

Save only critical messages while the client is disconnected by only indicating durability for critical subscriptions and CQs. When the client is connected to its servers, it receives messages for all keys and queries reqistered. When the client is disconnected, non-durable interest registrations and CQs are discontinued but all messages already in the queue for them remain there.

Note: For a single durable client ID, you must maintain the same durability of your registrations and queries between client runs.

Program the Client to Manage Durable Messaging

Program your durable client to be durable-messaging aware when it disconnects, reconnects, and handles events from the server.

1. Disconnect with a request to keep your queues active by using `Pool.close` or `ClientCache.close` with the boolean `keepalive` parameter.

   ```
   clientCache.close(true);
   ```

   To be retained during client down time, durable continuous queries (CQs) must be executing at the time of disconnect.

2. Program your durable client's reconnection to:

   1. If desired, detect whether the previously registered subscription queue is available upon durable client reconnection and the count of pending events in the queue. Based on the results, you can then decide whether to receive the remaining events or close the cache if the number is too large.

      For example, for a client with only the default pool created:

      ```
      int pendingEvents = cache.getDefaultPool().getPendingEventCount();

      if (pendingEvents == -2) { // client connected for the first time  … // c
      ontinue
      }
      else if (pendingEvents == -1) { // client reconnected but after the timeo
      ut period
      ```

```
… // handle possible data loss
}
else { // pendingEvents >= 0
… // decide to invoke readyForEvents() or ClientCache::close(false)/pool.
destroy()
}
```

For a client with multiple pools:

```
int pendingEvents = 0;

int pendingEvents1 = PoolManager.find("pool1").getPendingEventCount();

pendingEvents += (pendingEvents1 > 0) ? pendingEvents1 : 0;

int pendingEvents2 = PoolManager.find("pool2").getPendingEventCount();

pendingEvents += (pendingEvents2 > 0) ? pendingEvents2 : 0;

// process individual pool counts separately.
```

The `getPendingEventCount` API can return the following possible values: - A value representing a count of events pending at the server. Note that this count is an approximate value based on the time the durable client pool connected or reconnected to the server. Any number of invocations will return the same value. - A zero value if there are no events pending at server for this client pool - A negative value indicates that no queue is available at the server for the client pool. - -1 indicates that the client pool has reconnected to the server after its durable-client-timeout period has elapsed. The pool's subscription queue has been removed possibly causing data loss. - A value of -2 indicates that this client pool has connected to server for the first time.

2. Connect, initialize the client cache, regions, and any cache listeners, and create and execute any durable continuous queries.

3. Run all interest registration calls.

   Note: Registering interest with `InterestResultPolicy.KEYS_VALUES` initializes the client cache with the *current* values of specified keys. If concurrency checking is enabled for the region, any earlier (older) region events that are replayed to the client are ignored and are not sent to configured listeners. If your client must process all replayed events for a region, register with `InterestResultPolicy.KEYS` or `InterestResultPolicy.NONE` when reconnecting. Or, deactivate concurrency checking for the region in the client cache. See Consistency for Region Updates.

4. Call `ClientCache.readyForEvents` so the server will replay stored events. If the ready message is sent earlier, the client may lose events.

```
ClientCache clientCache = ClientCacheFactory.create();
// Here, create regions, listeners that are not defined in the cache.xml . . .
// Here, run all register interest calls before doing anything else
clientCache.readyForEvents();
```

3. When you program your durable client `CacheListener`:

1. Implement the callback methods to behave properly when stored events are replayed. The durable client's `CacheListener` must be able to handle having events played after the fact. Generally listeners receive events very close to when they happen, but the durable client may receive events that occurred minutes before and are not relevant to current cache state.
2. Consider whether to use the `CacheListener` callback method, `afterRegionLive`, which is provided specifically for the end of durable event replay. You can use it to perform application-specific operations before resuming normal event handling. If you do not wish to use this callback, and your listener is an instance of `CacheListener` (instead of a `CacheListenerAdapter`) implement `afterRegionLive` as an empty method.

Initial Operation

The initial startup of a durable client is similar to the startup of any other client, except that it specifically calls the `ClientCache.readyForEvents` method when all regions and listeners on the client are ready to process messages from the server.

Disconnection

While the client and servers are disconnected, their operation varies depending on the circumstances.

- **Normal disconnect**. When a client closes its connection, the servers stop sending messages to the client and release its connection. If the client requests it, the servers maintain the queues and durable interest list information until the client reconnects or times out. The non-durable interest lists are discarded. The servers continue to queue up incoming messages for entries on the durable interest list. All messages that were in the queue when the client disconnected remain in the queue. If the client requests not to have its subscriptions maintained, or if there are no durable subscriptions, the servers unregister the client and do the same cleanup as for a non-durable client.
- **Abnormal disconnect**. If the client crashes or loses its connections to all servers, the servers automatically maintain its message queue and durable subscriptions until it reconnects or times out.
- **Client disconnected but operational**. If the client operates while it is disconnected, it gets what data it can from the local client cache. Since updates are not allowed, the data can become stale. An `UnconnectedException` occurs if an update is attempted.
- **Client stays disconnected past timeout period**. The servers track how long to keep a durable subscription queue alive based on the `durable-client-timeout` setting. If the client remains disconnected longer than the timeout, the servers unregister the client and do the same cleanup that is performed for a non-durable client. The servers also log an alert. When a timed-out client reconnects, the servers treat it as a new client making its initial connection.

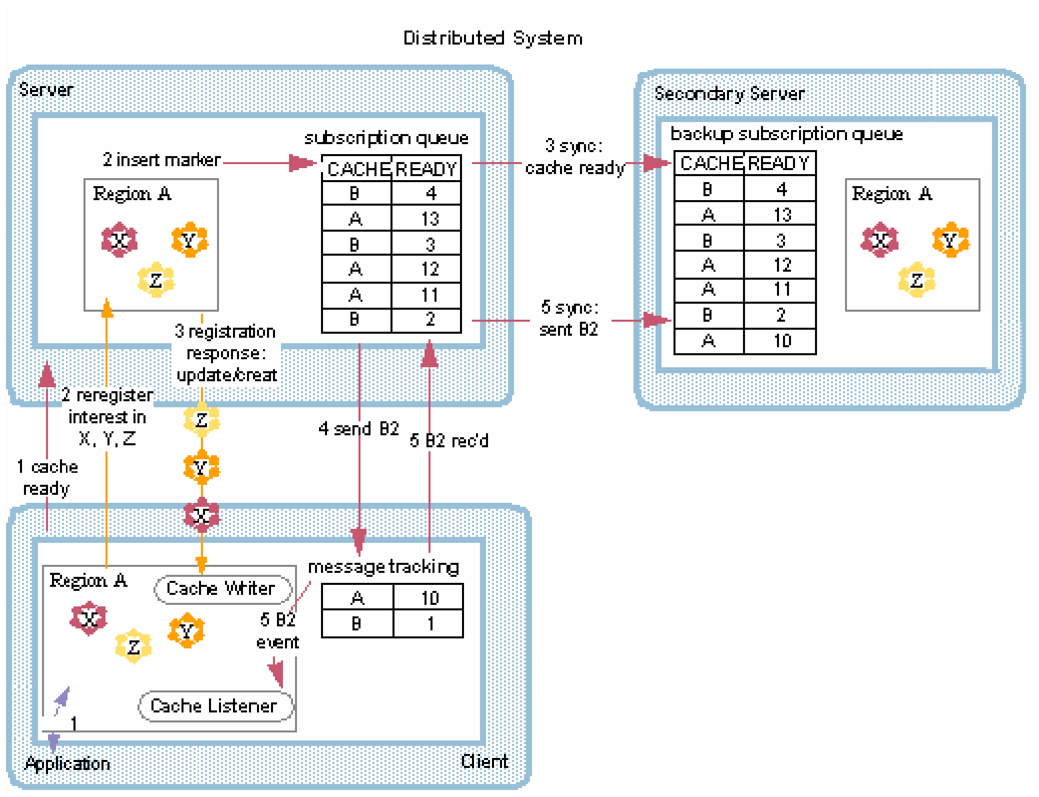
Reconnection

During initialization, the client cache is not blocked from doing operations, so you might be receiving old stored events from the server at the same time that your client cache is being updated by much more current events. These are the things that can act on the cache concurrently:

- Results returned by the server in response to the client's interest registrations.
- Client cache operations by the application.
- Callbacks triggered by replaying old events from the queue

VMware GemFire handles the conflicts between the application and interest registrations so they do not create cache update conflicts. But you must program your event handlers so they don't conflict with current operations. This is true for all event handlers, but it is especially important for those used in durable clients. Your handlers may receive events well after the fact and you must ensure your programming takes that into account.

This figure shows the three concurrent procedures during the initialization process. The application begins operations immediately on the client (step 1), while the client's cache ready message (also step 1) triggers a series of queue operations on the servers (starting with step 2 on the primary server). At the same time, the client registers interest (step 2 on the client) and receives a response from the server. Message B2 applies to an entry in Region A, so the cache listener handles B2's event. Because B2 comes before the marker, the client does not apply the update to the cache.



subscription queue

CACHE	READY
B	4
A	13
B	3
A	12
A	11
B	2

backup subscription queue

CACHE	READY
B	4
A	13
B	3
A	12
A	11
B	2
A	10

message tracking

A	10
B	1

Durable Event Replay

When a durable client reconnects before the timeout period, the servers replay the events that were stored while the client was gone and then resume normal event messaging to the client. To avoid overwriting current entries with old data, the stored events are not applied to the client cache. Stored events are distinguished from new normal events by a marker that is sent to the client once all old events are replayed.

1. All servers with a queue for this client place a marker in their queue when the client reconnects.
2. The primary server sends the queued messages to the client up to the marker.
3. The client receives the messages but does not apply the usual automatic updates to its cache. If cache listeners are installed, they handle the events.
4. The client receives the marker message indicating that all past events have been played back.
5. The server sends the current list of live regions.
6. For every `CacheListener` in each live region on the client, the marker event triggers the `afterRegionLive` callback. After the callback, the client begins normal processing of events from the server and applies the updates to its cache.

Even when a new client starts up for the first time, the client cache ready markers are inserted in the queues. If messages start coming into the new queues before the servers insert the marker, those messages are considered as having happened while the client was disconnected, and their events are replayed the same as in the reconnect case.

Application Operations During Interest Registration

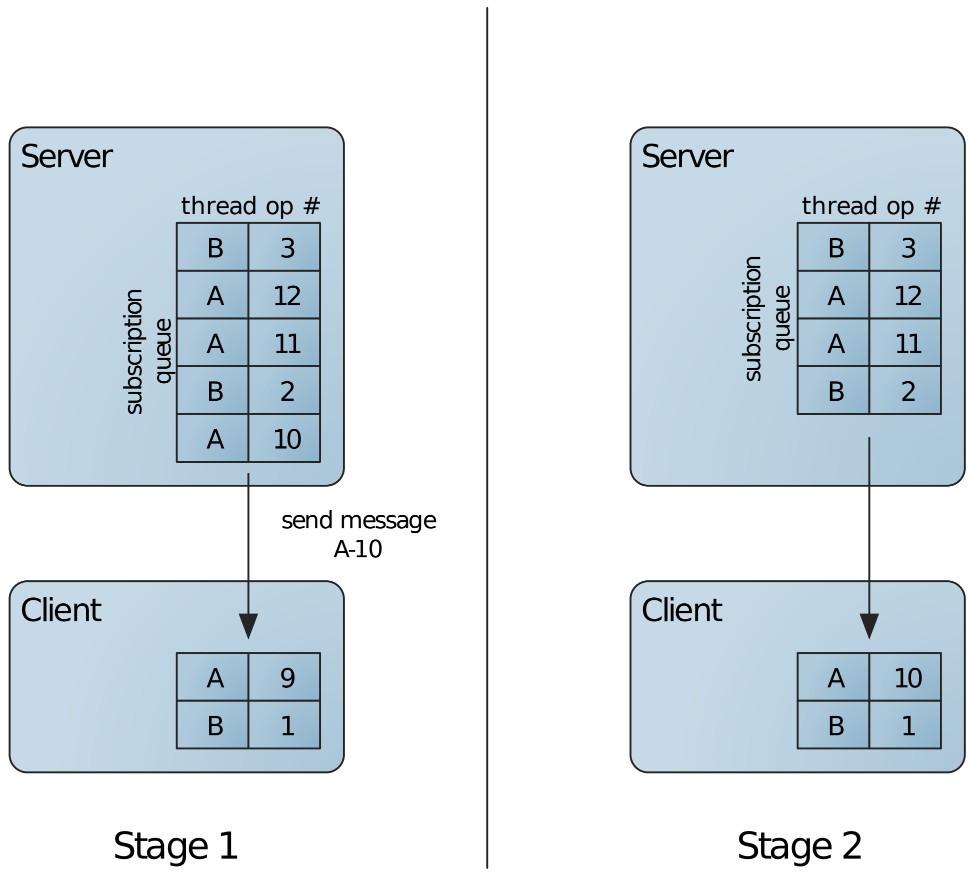
Application operations take precedence over interest registration responses. The client can perform operations while it is receiving its interest registration responses. When adding register interest responses to the client cache, the following rules are applied:

- If the entry already exists in the cache with a valid value, it is not updated.
- If the entry is invalid, and the register interest response is valid, the valid value is put into the cache.
- If an entry is marked destroyed, it is not updated. Destroyed entries are removed from the system after the register interest response is completed.
- If the interest response does not contain any results, because all of those keys are absent from the server's cache, the client's cache can start out empty. If the queue contains old messages related to those keys, the events are still replayed in the client's cache.

Tuning Client/Server Event Messaging

The server uses an asynchronous messaging queue to send events to its clients. Every event in the queue originates in an operation performed by a thread in a client, a server, or an application in the server's or some other cluster. The event message has a unique identifier composed of the originating thread's ID combined with its member's distributed system member ID, and the sequential ID of the operation. So the event messages originating in any single thread can be grouped and ordered by time from lowest sequence ID to highest. Servers and clients track the highest sequential ID for each member thread ID.

A single client thread receives and processes messages from the server, tracking received messages to make sure it does not process duplicate sends. It does this using the process IDs from originating threads.



The client's message tracking list holds the highest sequence ID of any message received for each originating thread. The list can become quite large in systems where there are many different threads coming and going and doing work on the cache. After a thread dies, its tracking entry is not needed. To avoid maintaining tracking information for threads that have died, the client expires entries that have had no activity for more than the `subscription-message-tracking-timeout`.

Conflate the Server Subscription Queue

Conflating the server subscription queue can save space in the server and time in message processing.

Enable conflation at the server level in the server region configuration:

```
<region ... >
  <region-attributes enable-subscription-conflation="true" />
</region>
```

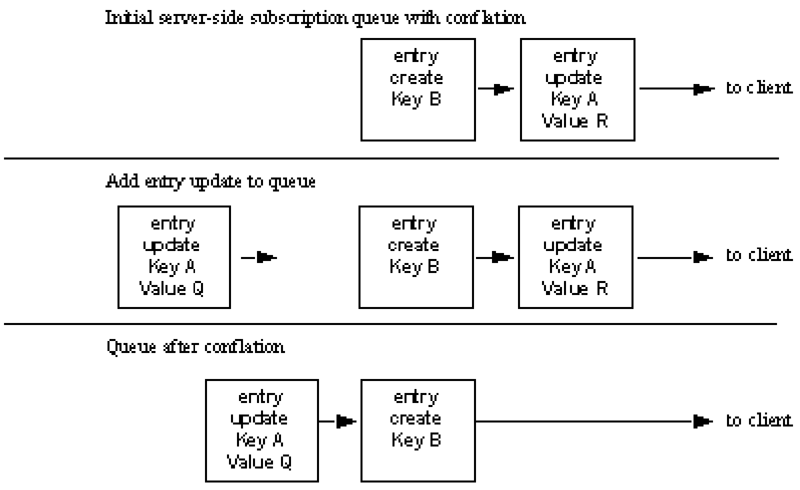
Override the server setting as needed, on a per-client basis, in the client's `gemfire.properties`:

```
conflate-events=false
```

Valid `conflate-events` settings are: - `server`, which uses the server settings - `true`, which conflates everything sent to the client - `false`, which does not conflate anything sent to this client

Conflation can both improve performance and reduce the amount of memory required on the server for queuing. The client receives only the latest available update in the queue for a particular entry key. Conflation is deactivated by default.

Conflation is particularly useful when a single entry is updated often and the intermediate updates don't require processing by the client. With conflation, if an entry is updated and there is already an update in the queue for its key, the existing update is removed and the new update is placed at the end of the queue. Conflation is only done on messages that are not in the process of being sent to the client.


Initial server-side subscription queue with conflation
entry create Key B
entry update Key A Value R
to client
Add entry update to queue
entry update Key A Value Q
entry create Key B
entry update Key A Value R
to client
Queue after conflation
entry update Key A Value Q
entry create Key B
to client

Note: This method of conflation is different from the one used for multi-site gateway sender queue conflation. It is the same as the method used for the conflation of peer-to-peer distribution messages within a single cluster.

Limit the Server's Subscription Queue Memory Use

These are options for limiting the amount of server memory the subscription queues consume.

- Optional: Conflate the subscription queue messages.
- Optional: Increase the frequency of queue synchronization by decreasing the pool's configuration parameter `subscription-ack-interval`. This applies only to configurations where server redundancy is used for high availability. Example:

  ```
  <!-- Set subscription ack interval to 3 seconds -->
  <cache>
    <pool ... subscription-enabled="true"
              subscription-ack-interval="3000">
    ...
  </pool>
  ```

  The client periodically sends an acknowledgment (`ack`) message to the server. Each message acknowledges the receipt of many events by the client. Since the server must retain every outbound event in the queue until its receipt is acknowledged, shortening the acknowledgment delay can reduce the average queue size, reducing the amount of server memory used for queueing.
- Optional: Limit Queue Size. Cap the server queue size using overflow or blocking. These options help avoid out of memory errors on the server in the case of slow clients. A slow

client slows the rate that the server can send messages, causing messages to back up in the queue, possibly leading to out of memory on the server. You can use one or the other of these options, but not both:

- Optional: Overflow to Disk. Configure subscription queue overflow by setting the server's `client-subscription` properties. With overflow, the most recently used (MRU) events are written out to disk, keeping the oldest events, the ones that are next in line to be sent to the client, available in memory. Example:

  ```
  <!-- Set overflow after 10K messages are enqueued -->
  <cache-server port="40404">
    <client-subscription
      eviction-policy="entry"
      capacity="10000"
      disk-store-name="svrOverflow"/>
  </cache-server>
  ```

- Optional: Block While Queue Full. Set the server's `maximum-message-count` to the maximum number of event messages allowed in any single subscription queue before incoming messages are blocked. You can only limit the message count, not the size allocated for messages. Examples:

  XML:

  ```
  <!-- Set the maximum message count to 50000 entries -->
    <cache-server port="41414" maximum-message-count="50000" />
  ```

  API:

  ```
  Cache cache = ...;
  CacheServer cacheServer = cache.addCacheServer();
  cacheServer.setPort(41414);
  cacheServer.setMaximumMessageCount(50000);
  cacheServer.start();
  ```

  Note: With this setting, one slow client can slow the server and all of its other clients because this blocks the threads that write to the queues. All operations that add messages to the queue block until the queue size drops to an acceptable level. If the regions feeding these queues are partitioned or have `distributed-ack` or `global` scope, operations on them remain blocked until their event messages can be added to the queue. If you are using this option and see stalling on your server region operations, your queue capacity might be too low for your application behavior.

Tune the Client's Subscription Message Tracking Timeout

If the client pool's `subscription-message-tracking-timeout` is set too low, your client will discard tracking records for live threads, increasing the likelihood of processing duplicate events from those threads.

This setting is especially important in systems where it is vital to avoid or greatly minimize duplicate events. If you detect that duplicate messages are being processed by your clients, increasing the timeout may help. Setting `subscription-message-tracking-timeout` may not completely eliminate duplicate entries, but careful configuration can help minimize occurrences.

Duplicates are monitored by keeping track of message sequence IDs from the source thread where the operation originated. For a long-running system, you would not want to track this information for very long periods or the information may be kept long enough for a thread ID to be recycled. If this happens, messages from a new thread may be discarded mistakenly as duplicates of messages from an old thread with the same ID. In addition, maintaining this tracking information for old threads uses memory that might be freed up for other things.

To minimize duplicates and reduce the size of the message tracking list, set your client `subscription-message-tracking-timeout` higher than double the sum of these times:

- The longest time your originating threads might wait between operations
- For redundant servers add:
    - The server's `message-sync-interval`
    - Total time required for failover (usually 7-10 seconds, including the time to detect failure)

You risk losing live thread tracking records if you set the value lower than this. This could result in your client processing duplicate event messages into its cache for the associated threads. It is worth working to set the `subscription-message-tracking-timeout` as low as you reasonably can.

```
<!-- Set the tracking timeout to 70 seconds -->
<pool name="client" subscription-enabled="true" subscription-message-tracking-timeout
="70000">
    ...
</pool>
```

Configuring Multi-Site (WAN) Event Queues

In a multi-site (WAN) installation, VMware GemFire uses gateway sender queues to distribute events for regions that are configured with a gateway sender. AsyncEventListeners also use an asynchronous event queue to distribute events for configured regions. This section describes additional options for configuring the event queues that are used by gateway senders or AsyncEventListener implementations.

Before you begin, set up your multi-site (WAN) installation or configure asynchronous event queues and AsyncEventListener implementations. See Configuring a Multi-site (WAN) System or Implementing an AsyncEventListener for Write-Behind Cache Event Handling.

- **Persisting an Event Queue**

  You can configure a gateway sender queue or an asynchronous event queue to persist data to disk similar to the way in which replicated regions are persisted.

- **Configuring Dispatcher Threads and Order Policy for Event Distribution**

  By default, VMware GemFire uses multiple dispatcher threads to process region events simultaneously in a gateway sender queue for distribution between sites, or in an asynchronous event queue for distributing events for write-behind caching. With serial queues, you can also configure the ordering policy for dispatching those events.

- **Conflating Events in a Queue**

  Conflating a queue improves distribution performance. When conflation is enabled, only the latest queued value is sent for a particular key.

Persisting an Event Queue

You can configure a gateway sender queue or an asynchronous event queue to persist data to disk similar to the way in which replicated regions are persisted.

Persisting a queue provides high availability for the event messaging that the sender performs. For example, if a persistent gateway sender queue exits for any reason, when the member that hosts the sender restarts it automatically reloads the queue and resumes sending messages. If an asynchronous event queue exits for any reason, write-back caching can resume where it left off when the queue is brought back online. VMware GemFire persists an event queue if you set the `enable-persistence` attribute to true. The queue is persisted to the disk store specified in the queue's `disk-store-name` attribute, or to the default disk store if you do not specify a store name.

You must configure the event queue to use persistence if you are using persistent regions. The use of non-persistent event queues with persistent regions is not supported.

When you enable persistence for a queue, the `maximum-queue-memory` attribute determines how much memory the queue can consume before it overflows to disk. By default, this value is set to 100MB.

Note: If you configure a parallel queue and/or you configure multiple dispatcher threads for a queue, the values that are defined in the `maximum-queue-memory` and `disk-store-name` attributes apply to each instance of the queue.

In the example below the gateway sender queue uses "diskStoreA" for persistence and overflow, and the queue has a maximum queue memory of 100MB:

- XML example:

```
<cache>
  <gateway-sender id="persistedsender1" parallel="false"
   remote-distributed-system-id="1"
   enable-persistence="true"
   disk-store-name="diskStoreA"
   maximum-queue-memory="100"/>
   ...
</cache>
```

- API example:

```
Cache cache = new CacheFactory().create();

GatewaySenderFactory gateway = cache.createGatewaySenderFactory();
gateway.setParallel(false);
gateway.setPersistenceEnabled(true);
gateway.setDiskStoreName("diskStoreA");
gateway.setMaximumQueueMemory(100);
GatewaySender sender = gateway.create("persistedsender1", "1");
sender.start();
```

- gfsh:

```
gfsh>create gateway-sender --id="persistedsender1 --parallel=false
--remote-distributed-system-id=1 --enable-persistence=true --disk-store-name=di
skStoreA
--maximum-queue-memory=100
```

If you were to configure 10 dispatcher threads for the serial gateway sender, then the total maximum memory for the gateway sender queue would be 1000MB on each VMware GemFire member that hosted the sender, because VMware GemFire creates a separate copy of the queue per thread..

The following example shows a similar configuration for an asynchronous event queue:

- XML example:

```
<cache>
   <async-event-queue id="persistentAsyncQueue" persistent="true"
    disk-store-name="diskStoreA" parallel="true">
      <async-event-listener>
         <class-name>MyAsyncEventListener</class-name>
         <parameter name="url">
           <string>jdbc:db2:SAMPLE</string>
         </parameter>
         <parameter name="username">
           <string>gfeadmin</string>
         </parameter>
         <parameter name="password">
           <string>admin1</string>
         </parameter>
      </async-event-listener>
    </async-event-queue>
...
</cache>
```

- API example:

```
Cache cache = new CacheFactory().create();
AsyncEventQueueFactory factory = cache.createAsyncEventQueueFactory();
factory.setPersistent(true);
factory.setDiskStoreName("diskStoreA");
factory.setParallel(true);
AsyncEventListener listener = new MyAsyncEventListener();
AsyncEventQueue persistentAsyncQueue = factory.create("customerWB", listener);
```

- gfsh:

```
gfsh>create async-event-queue --id="persistentAsyncQueue" --persistent=true
--disk-store="diskStoreA" --parallel=true --listener=MyAsyncEventListener
--listener-param=url#jdbc:db2:SAMPLE --listener-param=username#gfeadmin --liste
ner-param=password#admin1
```

Configuring Dispatcher Threads and Order Policy for Event Distribution

By default, VMware GemFire uses multiple dispatcher threads to process region events simultaneously in a gateway sender queue for distribution between sites, or in an asynchronous event queue for distributing events for write-behind caching. With serial queues, you can also configure the ordering policy for dispatching those events.

By default, a gateway sender queue or asynchronous event queue uses 5 dispatcher threads per queue. This provides support for applications that have the ability to process queued events concurrently for distribution to another VMware GemFire site or listener. If your application does not require concurrent distribution, or if you do not have enough resources to support the requirements of multiple dispatcher threads, then you can configure a single dispatcher thread to process a queue.

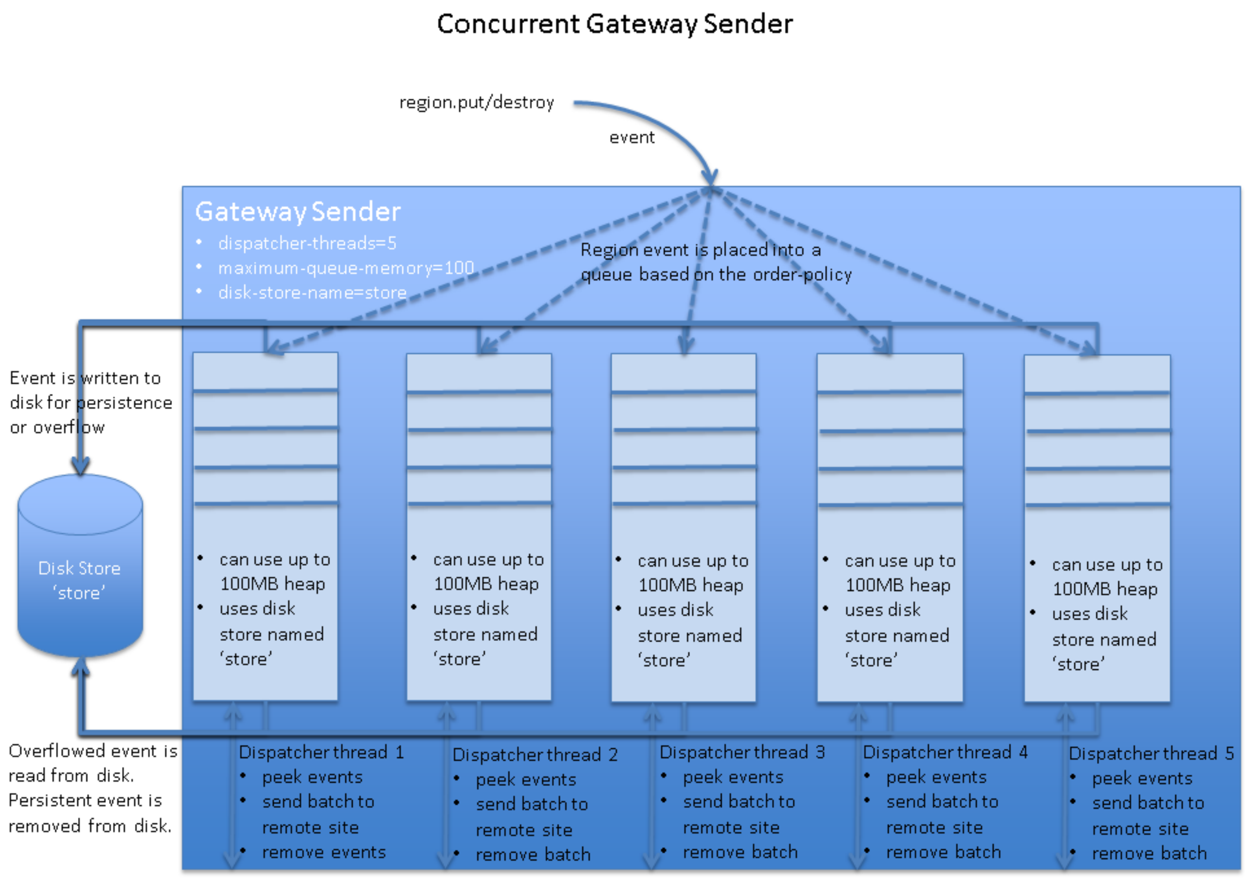
- Using Multiple Dispatcher Threads to Process a Queue
- Performance and Memory Considerations
- Configuring the Ordering Policy for Serial Queues
- Examples—Configuring Dispatcher Threads and Ordering Policy for a Serial Gateway Sender Queue

Using Multiple Dispatcher Threads to Process a Queue

When multiple dispatcher threads are configured for a parallel queue, VMware GemFire simply uses multiple threads to process the contents of each individual queue. The total number of queues that are created is still determined by the number of VMware GemFire members that host the region.

When multiple dispatcher threads are configured for a serial queue, VMware GemFire creates an additional copy of the queue for each thread on each member that hosts the queue. To obtain the maximum throughput, increase the number of dispatcher threads until your network is saturated.

The following diagram illustrates a serial gateway sender queue that is configured with multiple dispatcher threads.



Performance and Memory Considerations

When a serial gateway sender or an asynchronous event queue uses multiple dispatcher threads, consider the following:

- Queue attributes are repeated for each copy of the queue that is created for a dispatcher thread. That is, each concurrent queue points to the same disk store, so the same disk directories are used. If persistence is enabled and overflow occurs, the threads that insert entries into the queues compete for the disk. This applies to application threads and dispatcher threads, so it can affect application performance.
- The `maximum-queue-memory` setting applies to each copy of the serial queue. If you configure 10 dispatcher threads and the maximum queue memory is set to 100MB, then the total maximum queue memory for the queue is 1000MB on each member that hosts the queue.

Configuring the Ordering Policy for Serial Queues

When using multiple `dispatcher-threads` (greater than 1) with a serial event queue, you can also configure the `order-policy` that those threads use to distribute events from the queue. The valid order policy values are:

- **key (default)**. All updates to the same key are distributed in order. VMware GemFire preserves key ordering by placing all updates to the same key in the same dispatcher thread queue. You typically use key ordering when updates to entries have no relationship to each other, such as for an application that uses a single feeder to distribute stock updates to several other systems.

- **thread**. All region updates from a given thread are distributed in order. VMware GemFire preserves thread ordering by placing all region updates from the same thread into the same dispatcher thread queue. In general, use thread ordering when updates to one region entry affect updates to another region entry.
- **partition**. All region events that share the same partitioning key are distributed in order. Specify partition ordering when applications use a PartitionResolver to implement custom partitioning. With partition ordering, all entries that share the same "partitioning key" (RoutingObject) are placed into the same dispatcher thread queue.

You cannot configure the `order-policy` for a parallel event queue, because parallel queues cannot preserve event ordering for regions. Only the ordering of events for a given partition (or in a given queue of a distributed region) can be preserved.

Examples—Configuring Dispatcher Threads and Ordering Policy for a Serial Gateway Sender Queue

To increase the number of dispatcher threads and set the ordering policy for a serial gateway sender, use one of the following mechanisms.

- **cache.xml configuration**

```
<cache>
  <gateway-sender id="NY" parallel="false"
   remote-distributed-system-id="1"
   enable-persistence="true"
   disk-store-name="gateway-disk-store"
   maximum-queue-memory="200"
   dispatcher-threads=7 order-policy="key"/>
   ...
</cache>
```

- **Java API configuration**

```
Cache cache = new CacheFactory().create();

GatewaySenderFactory gateway = cache.createGatewaySenderFactory();
gateway.setParallel(false);
gateway.setPersistenceEnabled(true);
gateway.setDiskStoreName("gateway-disk-store");
gateway.setMaximumQueueMemory(200);
gateway.setDispatcherThreads(7);
gateway.setOrderPolicy(OrderPolicy.KEY);
GatewaySender sender = gateway.create("NY", "1");
sender.start();
```

- **gfsh:**

```
gfsh>create gateway-sender -d="NY"
   --parallel=false
   --remote-distributed-system-id="1"
   --enable-persistence=true
   --disk-store-name="gateway-disk-store"
   --maximum-queue-memory=200
   --dispatcher-threads=7
   --order-policy="key"
```

The following examples show how to set dispatcher threads and ordering policy for an asynchronous event queue:

- **cache.xml configuration**

```
<cache>
   <async-event-queue id="sampleQueue" persistent="true"
    disk-store-name="async-disk-store" parallel="false"
    dispatcher-threads=7 order-policy="key">
      <async-event-listener>
         <class-name>MyAsyncEventListener</class-name>
         <parameter name="url">
           <string>jdbc:db2:SAMPLE</string>
         </parameter>
         <parameter name="username">
           <string>gfeadmin</string>
         </parameter>
         <parameter name="password">
           <string>admin1</string>
         </parameter>
    </async-event-listener>
    </async-event-queue>
...
</cache>
```

- **Java API configuration**

```
Cache cache = new CacheFactory().create();
AsyncEventQueueFactory factory = cache.createAsyncEventQueueFactory();
factory.setPersistent(true);
factory.setDiskStoreName("async-disk-store");
factory.setParallel(false);
factory.setDispatcherThreads(7);
factory.setOrderPolicy(OrderPolicy.KEY);
AsyncEventListener listener = new MyAsyncEventListener();
AsyncEventQueue sampleQueue = factory.create("customerWB", listener);
```

  Entry updates in the current, in-process batch are not eligible for conflation.

- **gfsh:**

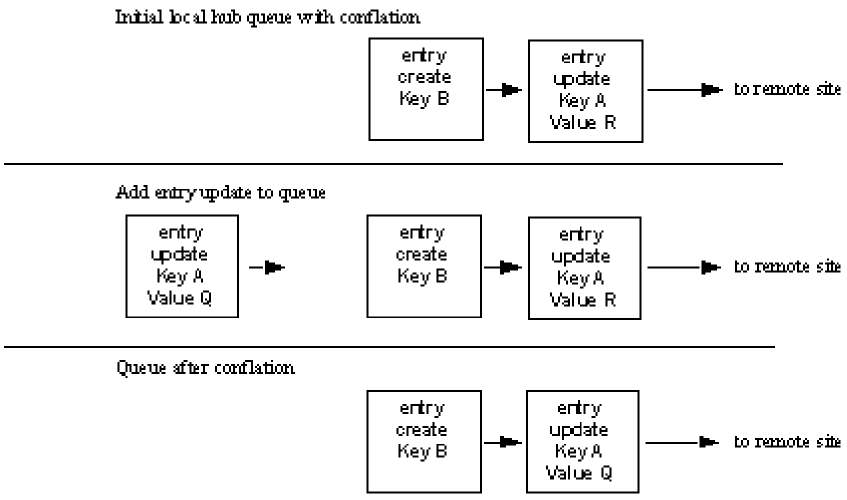
```
gfsh>create async-event-queue --id="sampleQueue" --persistent=true
--disk-store="async-disk-store" --parallel=false
--dispatcher-threads=7 order-policy="key"
--listener=myAsycEventListener
--listener-param=url#jdbc:db2:SAMPLE
--listener-param=username#gfeadmin
--listener-param=password#admin1
```

Conflating Events in a Queue

Conflating a queue improves distribution performance. When conflation is enabled, only the latest queued value is sent for a particular key.

Note: Do not use conflation if your receiving applications depend on the specific ordering of entry modifications, or if they need to be notified of every change to an entry.

Conflation is most useful when a single entry is updated frequently, but other sites only need to know the current value of the entry (rather than the value of each update). When an update is added to a queue that has conflation enabled, if there is already an update message in the queue for the entry key, then the existing message assumes the value of the new update and the new update is dropped, as shown here for key A.


Initial local hub queue with conflation
entry create Key B
entry update Key A Value R
to remote site
Add entry update to queue
entry update Key A Value Q
entry create Key B
entry update Key A Value R
to remote site
Queue after conflation
entry create Key B
entry update Key A Value Q
to remote site

Note: This method of conflation is different from the one used for server-to-client subscription queue conflation and peer-to-peer distribution within a cluster.

Examples—Configuring Conflation for a Gateway Sender Queue

To enable conflation for a gateway sender queue, use one of the following mechanisms:

- **cache.xml configuration**

```
<cache>
  <gateway-sender id="NY" parallel="true"
   remote-distributed-system-id="1"
   enable-persistence="true"
   disk-store-name="gateway-disk-store"
   enable-batch-conflation="true"/>
   ...
</cache>
```

- **Java API configuration**

```
Cache cache = new CacheFactory().create();

GatewaySenderFactory gateway = cache.createGatewaySenderFactory();
gateway.setParallel(true);
gateway.setPersistenceEnabled(true);
gateway.setDiskStoreName("gateway-disk-store");
gateway.setBatchConflationEnabled(true);
```

```
GatewaySender sender = gateway.create("NY", "1");
sender.start();
```

Entry updates in the current, in-process batch are not eligible for conflation.

- **gfsh:**

```
gfsh>create gateway-sender --id="NY" --parallel=true
   --remote-distributed-system-id="1"
   --enable-persistence=true
   --disk-store-name="gateway-disk-store"
   --enable-batch-conflation=true
```

The following examples show how to configure conflation for an asynchronous event queue:

- **cache.xml configuration**

```
<cache>
   <async-event-queue id="sampleQueue" persistent="true"
    disk-store-name="async-disk-store" parallel="false"
    enable-batch-conflation="true">
      <async-event-listener>
         <class-name>MyAsyncEventListener</class-name>
         <parameter name="url">
           <string>jdbc:db2:SAMPLE</string>
         </parameter>
         <parameter name="username">
           <string>gfeadmin</string>
         </parameter>
         <parameter name="password">
           <string>admin1</string>
         </parameter>
   </async-event-listener>
 </async-event-queue>
...
</cache>
```

- **Java API configuration**

```
Cache cache = new CacheFactory().create();
AsyncEventQueueFactory factory = cache.createAsyncEventQueueFactory();
factory.setPersistent(true);
factory.setDiskStoreName("async-disk-store");
factory.setParallel(false);
factory.setBatchConflationEnabled(true);
AsyncEventListener listener = new MyAsyncEventListener();
AsyncEventQueue sampleQueue = factory.create("customerWB", listener);
```

Entry updates in the current, in-process batch are not eligible for conflation.

- **gfsh:**

```
gfsh>create async-event-queue --id="sampleQueue" --persistent=true
--disk-store="async-disk-store" --parallel="false"
--listener=myAsyncEventListener
--listener-param=url#jdbc:db2:SAMPLE
```

```
--listener-param=username#gfeadmin
--listener-param=password#admin1
```

Delta Propagation

Delta propagation allows you to reduce the amount of data you send over the network by including only changes to objects rather than the entire object.

- **How Delta Propagation Works**

  Delta propagation reduces the amount of data you send over the network. You do this by only sending the change, or delta, information about an object, instead of sending the entire changed object. If you do not use cloning when applying the deltas, you can also expect to generate less garbage in your receiving JVMs.

- **When to Avoid Delta Propagation**

  Generally, the larger your objects and the smaller the deltas, the greater the benefits of using delta propagation. Partitioned regions with higher redundancy levels generally benefit more from delta propagation. However, in some application scenarios, delta propagation does not show any significant benefits. On occasion it results in performance degradation.

- **Delta Propagation Properties**

  This topic describes the properties that can be used to configure delta propagation.

- **Implementing Delta Propagation**

  By default, delta propagation is enabled in your cluster. When enabled, delta propagation is used for objects that implement `org.apache.geode.Delta`. You program the methods to store and extract delta information for your entries and to apply received delta information.

- **Errors In Delta Propagation**

  This topic lists the errors that can occur when using delta propagation.

- **Delta Propagation Example**

  This topic provides an example of delta propagation.

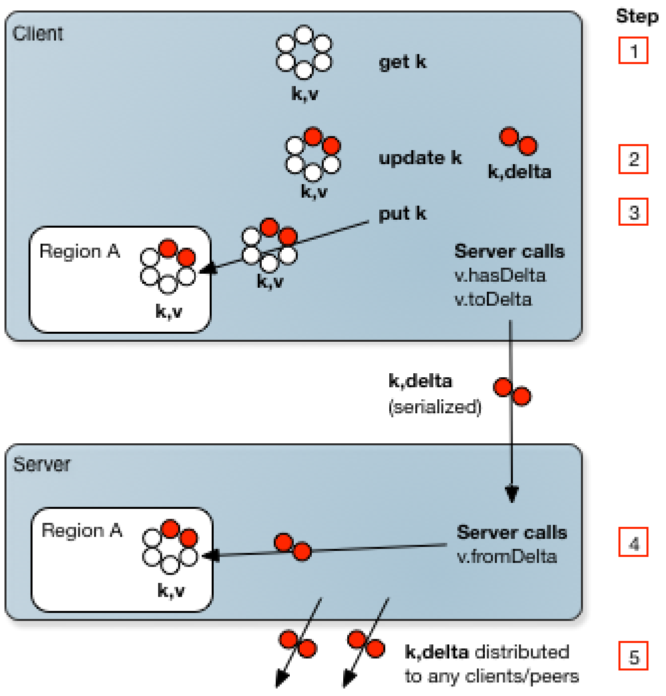
How Delta Propagation Works

Delta propagation reduces the amount of data you send over the network. You do this by only sending the change, or delta, information about an object, instead of sending the entire changed object. If you do not use cloning when applying the deltas, you can also expect to generate less garbage in your receiving JVMs.

In most distributed data management systems, the data stored in the system tends to be created once and then updated frequently. These updates are sent to other members for event propagation, redundancy management, and cache consistency in general. Tracking only the changes in an updated object and sending only the deltas mean lower network transmission costs and lower object serialization/deserialization costs. Performance improvements can be significant, especially when changes to an object are small relative to its overall size.

VMware GemFire propagates object deltas using methods that you program. The methods are in the `Delta` interface, which you implement in your cached objects' classes. If any of your classes are plain old Java objects, you need to wrap them for this implementation.

This figure shows delta propagation for a change to an entry with key, k, and value object, v.



1. `get` **operation**. The `get` works as usual: the cache returns the full entry object from the local cache or, if it isn't available there, from a remote cache or from a loader.
2. **update methods**. You need to add code to the object's update methods so that they save delta information for object updates, in addition to the work they were already doing.
3. `put` **operation**. The `put` works as usual in the local cache, using the full value, then calls `hasDelta` to see if there are deltas and `toDelta` to serialize the information. Distribution is the same as for full values, according to member and region configuration.
4. **receipt of delta at remote member**. `fromDelta` extracts the delta information that was serialized by `toDelta` and applies it to the object in the local cache. The delta is applied directly to the existing value or to a clone, depending on how you configure it for the region.
5. **additional distributions**. As with full distributions, receiving members forward the delta according to their configurations and connections to other members. For example, if VM1 is a client and VM2 is a server, VM2 forwards the delta to its peers and its other clients as needed. Receiving members do not recreate the delta; `toDelta` is only called in the originating member.

General Characteristics of Delta Propagation

To use the delta propagation feature, all updates on a key in a region must have value types that implement the `Delta` interface. You cannot mix object types for an entry key where some of the types implement delta and some do not. This is because, when a type implementing the delta interface is received for an update, the existing value for the key is cast to a `Delta` type to apply the received delta. If the existing type does not also implement the `Delta` interface, the operation throws a `ClassCastException`.

Note: Only the object itself being placed in the cache can implement the `Delta` interface and propagate changes. Any sub-objects of the cache object do not propagate their changes.

Sometimes `fromDelta` cannot be invoked because there is no object to apply the delta to in the receiving cache. When this happens, the system automatically does a full value distribution to the receiver. These are the possible scenarios: 1. If the system can determine beforehand that the receiver does not have a local copy, it sends the initial message with the full value. This is possible when regions are configured with no local data storage, such as with the region shortcut settings `PARTITION_PROXY` and `REPLICATE_PROXY`. These configurations are used to accomplish things like provide data update information to listeners and to pass updates forward to clients. 2. In less obvious cases, such as when an entry has been locally deleted, first the delta is sent, then the receiver requests a full value and that is sent. Whenever the full value is received, any further distributions to the receiver's peers or clients uses the full value.

VMware GemFire also does not propagate deltas for:

- Transactional commit
- The `putAll` operation
- JVMs running VMware GemFire versions that do not support delta propagation (6.0 and earlier)

Supported Topologies and Limitations

The following topologies support delta propagation (with some limitations):

- **Peer-to-peer**. VMware GemFire system members distribute and receive entry changes using delta propagation, with these requirements and caveats:
  - Regions must be partitioned or have their scope set to `distributed-ack` or `global`. The region shortcut settings for distributed regions use `distributed-ack` `scope`. Delta propagation does not work for regions with `distributed-no-ack` `scope` because the receiver could not recover if an exception occurred while applying the delta.
  - For partitioned regions, if a receiving peer does not hold the primary or a secondary copy of the entry, but still requires a value, the system automatically sends the full value.
  - To receive deltas, a region must be non-empty. The system automatically sends the full value to empty regions. Empty regions can send deltas.
- **Client/server**. VMware GemFire clients can always send deltas to the servers, and servers can usually sent deltas to clients. These configurations require the servers to send full values to the clients, instead of deltas:
  - When the client's `gemfire.properties` setting `conflate-events` is set to true, the servers send full values for all regions.
  - When the server region attribute `enable-subscription-conflation` is set to true and the client `gemfire.properties` setting `conflate-events` is set to `server`, the servers send full values for the region.

    - When the client region is configured with the `PROXY` client region shortcut setting (empty client region), servers send full values.
- **Multi-site (WAN)**. Gateway senders do not send Deltas. The full value is always sent.

When to Avoid Delta Propagation

Generally, the larger your objects and the smaller the deltas, the greater the benefits of using delta propagation. Partitioned regions with higher redundancy levels generally benefit more from delta propagation. However, in some application scenarios, delta propagation does not show any significant benefits. On occasion it results in performance degradation.

By default, delta propagation is enabled in your cluster.

These are the main factors that can reduce the performance benefits of using delta propagation:

- The added costs of deserializing your objects to apply deltas. Applying a delta requires the entry value to be deserialized. Once this is done, the object is stored back in the cache in deserialized form. This aspect of delta propagation only negatively impacts your system if your objects are not already being deserialized for other reasons, such as for indexing and querying or for listener operations. Once stored in deserialized form, there are reserialization costs for operations that send the object outside of the member, like distribution from a gateway sender, values sent in response to `netSearch` or client requests, and storage to disk. The more operations that require reserialization, the higher the overhead of deserializing the object. As with all serialization efforts, you can improve performance in serialization and deserialization by providing custom implementations of `DataSerializable` for your objects.
- Cloning when applying the delta. Using cloning can affect performance and generates extra garbage. Not using cloning is risky however, as you are modifying cached values in place. Without cloning, make sure you synchronize your entry access to keep your cache from becoming inconsistent.
- Problems applying the delta that cause the system to go back to the originator for the full entry value. When this happens, the overall operation costs more than sending the full entry value in the first place. This can be additionally aggravated if your delta is sent to a number of recipients, all or most of them request a full value, and the full value send requires the object to be serialized.
- Disk I/O costs associated with overflow regions. If you use eviction with overflow to disk, on-disk values must be brought into memory in order to apply the delta. This is much more costly than just removing the reference to the disk copy, as you would do with a full value distribution into the cache.

Delta Propagation Properties

This topic describes the properties that can be used to configure delta propagation.

Delta propagation properties can be configured through the API and through the `gemfire.properties` and `cache.xml` files.

delta-propagation

A `gemfire.properties` boolean that enables or disables delta propagation. When false, full entry values are sent for every update. The default setting is true, which enables delta propagation.

Disable delta propagation as follows:

- `gemfire.properties`:

  ```
  delta-propagation=false
  ```

- API:

  ```
  Properties props = new Properties();
  props.setProperty("delta-propagation", false);
  this.cache = new ClientCacheFactory(props).create();
  ```

cloning-enabled

A region attributes boolean that affects how `fromDelta` applies deltas to the local cache. When true, the updates are applied to a clone of the value and then the clone is saved to the cache. When false, the value is modified in place in the cache. The default value is false.

Exceptions to this behavior:

- If the `Cache` attribute `copy-on-read` is true, cloning is enabled, regardless of what this attribute is set to.
- If the `Region` attribute `off-heap` is true, cloning is enabled, regardless of what this attribute is set to.

Cloning can be expensive, but it ensures that the new object is fully initialized with the delta before any application code sees it.

When cloning is enabled, by default VMware GemFire does a deep copy of the object, using serialization. You may be able to improve performance by implementing `java.lang.Cloneable` and then implementing the `clone` method, making a deep copy of anything to which a delta may be applied. The goal is to reduce significantly the overhead of copying the object while still retaining the isolation needed for your deltas.

Without cloning:

- It is possible for application code to read the entry value as it is being modified, possibly seeing the value in an intermediate, inconsistent state, with just part of the delta applied. You may choose to resolve this issue by having your application code synchronize on reads and writes.
- VMware GemFire loses any reference to the old value because the old value is transformed in place into the new value. Because of this, your `CacheListener` sees the same new value returned for `EntryEvent.getOldValue` and `EntryEvent.getNewValue` .
- Exceptions thrown from `fromDelta` may leave your cache in an inconsistent state. Without cloning, any interruption of the delta application could leave you with some of the fields in your cached object changed and others unchanged. If you do not use cloning, keep this in mind when you program your error handling in your `fromDelta` implementation.

With cloning:

- The `fromDelta` method generates more garbage in memory.
- Performance is reduced.

Enable cloning as follows:

- `cache.xml`:

  ```
  <region name="region_with_cloning">
      <region-attributes refid="REPLICATE" cloning-enabled="true">
      </region-attributes>
  </region>
  ```

- API:

  ```
  RegionFactory rf = cache.createRegionFactory(REPLICATE);
  rf.setCloningEnabled(true);
  custRegion = rf.create("customer");
  ```

- gfsh:

  ```
  gfsh>create region --name="region_with_cloning" --type=REPLICATE
  --enable-cloning=true
  ```

Implementing Delta Propagation

By default, delta propagation is enabled in your cluster. When enabled, delta propagation is used for objects that implement `org.apache.geode.Delta`. You program the methods to store and extract delta information for your entries and to apply received delta information.

Use the following procedure to implement delta propagation in your cluster.

1. Study your object types and expected application behavior to determine which regions can benefit from using delta propagation. Delta propagation does not improve performance for all data and data modification scenarios. See When to Avoid Delta Propagation.
2. For each region where you are using delta propagation, choose whether to enable cloning using the delta propagation property `cloning-enabled`. Cloning is disabled by default. See Delta Propagation Properties.
3. If you do not enable cloning, review all associated listener code for dependencies on `EntryEvent.getOldValue`. Without cloning, VMware GemFire modifies the entry in place and so loses its reference to the old value. For delta events, the `EntryEvent` methods `getOldValue` and `getNewValue` both return the new value.
4. For every class where you want delta propagation, implement `org.apache.geode.Delta` and update your methods to support delta propagation. Exactly how you do this depends on your application and object needs, but these steps describe the basic approach:
    1. If the class is a plain old Java object (POJO), wrap it for this implementation and update your code to work with the wrapper class.
    2. Define as transient any extra object fields that you use to manage delta state. This can help performance when the full object is distributed. Whenever standard Java serialization is used, the transient keyword indicates to Java to not serialize the field.

3. Study the object contents to decide how to handle delta changes. Delta propagation has the same issues of distributed concurrency control as the distribution of full objects, but on a more detailed level. Some parts of your objects may be able to change independent of one another while others may always need to change together. Send deltas large enough to keep your data logically consistent. If, for example, field A and field B depend on each other, then your delta distributions should either update both fields or neither. As with regular updates, the fewer producers you have on a data region, the lower your likelihood of concurrency issues.
4. In the application code that puts entries, put the fully populated object into the local cache. Even though you are planning to send only deltas, errors on the receiving end could cause VMware GemFire to request the full object, so you must provide it to the originating put method. Do this even in empty producers, with regions configured for no local data storage. This usually means doing a get on the entry unless you are sure it does not already exist anywhere in the distributed region.
5. Change each field's update method to record information about the update. The information must be sufficient for `toDelta` to encode the delta and any additional required delta information when it is invoked.
6. Write `hasDelta` to report on whether a delta is available.
7. Write `toDelta` to create a byte stream with the changes to the object and any other information `fromDelta` will need to apply the changes. Before returning from `toDelta`, reset your delta state to indicate that there are no delta changes waiting to be sent.
8. Write `fromDelta` to decode the byte stream that `toDelta` creates and update the object.
9. Make sure you provide adequate synchronization to your object to maintain a consistent object state. If you do not use cloning, you will probably need to synchronize on reads and writes to avoid reading partially written updates from the cache.This synchronization might involve `toDelta`, `fromDelta`, `toData`, `fromData`, and other methods that access or update the object. Additionally, your implementation should take into account the possibility of concurrent invocations of `fromDelta` and one or more of the object's update methods.

Errors In Delta Propagation

This topic lists the errors that can occur when using delta propagation.

Errors in delta propagation fall into two categories based on how they are handled by the system:

- Problems applying the delta that can be remedied by requesting the full value in place of the delta. Your `put` operation does not see errors or exceptions related to this type of delta propagation failure. The system automatically does a full value distribution from the sender to the receiver where the problem occurs. This type of error includes:
    - Unavailable entry value in the receiving cache, either because the entry is missing or its value is null. In both cases, there is nothing to apply the delta to and the full value must be sent. This is most likely to occur if you destroy or invalidate your

entries locally, either through application calls or through configured actions like eviction or entry expiration.

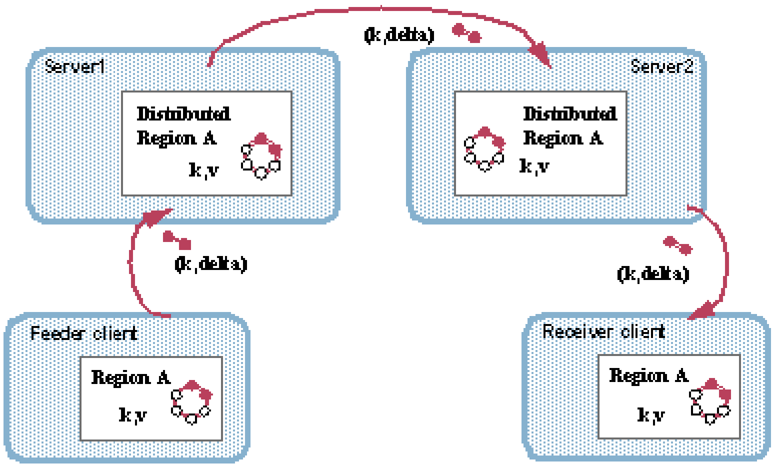
- `InvalidDeltaException` thrown by `fromDelta` method, programmed by you. This exception enables you to avoid applying deltas that would violate data consistency checks or other application requirements.
- Any error applying the delta in a client in server-to-client propagation. The client logs a warning in addition to retrieving the full value from the server.

- Problems creating or distributing the delta that cannot be fixed by distributing the full value. In these cases, your `put` operation fails with an exception. This type of error includes:
  - Error or exception in `hasDelta` or `toDelta`.
  - Error or exception in a server or peer receiver that fall outside of the situations described above in the first category.

Delta Propagation Example

This topic provides an example of delta propagation.

In this example, the feeder client is connected to the first server, and the receiver client is connected to the second. The servers are peers to each other.



The example demonstrates the following operations:

1. In the Feeder client, the application updates the entry object and puts the entry. In response to the `put`, VMware GemFire calls `hasDelta`, which returns true, so VMware GemFire calls `toDelta` and forwards the extracted delta to the server. If `hasDelta` returned false, VMware GemFire would distribute the full entry value.
2. In Server1, VMware GemFire applies the delta to the cache, distributes the received delta to the server's peers, and forwards it to any other clients with interest in the entry (there are no other clients to Server1 in this example)
3. In Server2, VMware GemFire applies the delta to the cache and forwards it to its interested clients, which in this case is just the Receiver client.

This example shows the basic approach to programming a `Delta` implementation.

```
package delta;
```

```
import org.apache.geode.Delta;
import org.apache.geode.InvalidDeltaException;
import java.io.DataInput;
import java.io.DataOutput;
import java.io.IOException;
import java.io.Serializable;

/**
 * Sample implementation of Delta
 *
 * @author GemStone Systems, Inc.
 * @since 6.1
 */
public class SimpleDelta implements Delta, Serializable {

// Original object fields
    private int intVal;
    private double doubleVal;

    // Added for delta - one boolean per field to track changed status
    private transient boolean intFldChd = false;
    private transient boolean dblFldChd = false;

    public SimpleDelta(){}

    public SimpleDelta(int i, double d){
        this.intVal = i;
        this.doubleVal = d;
    }

    public boolean hasDelta() {
        return this.intFldChd || this.dblFldChd;
    }

    public void toDelta(DataOutput out) throws IOException {
        System.out.println("Extracting delta from " + this.toString());
    // Write information on what has changed to the
    // data stream, so fromDelta knows what it's getting
        out.writeBoolean(intFldChd);
        if (intFldChd) {
    // Write just the changes into the data stream
            out.writeInt(this.intVal);
    // Once the delta information is written, reset the delta status field
            this.intFldChd = false;
            System.out.println(" Extracted delta from field 'intVal' = "
                                                    + this.intVal);
        }
        out.writeBoolean(dblFldChd);
        if (dblFldChd) {
            out.writeDouble(this.doubleVal);
            this.dblFldChd = false;
            System.out.println(" Extracted delta from field 'doubleVal' = "
                                                    + this.doubleVal);
        }
    }

    public void fromDelta(DataInput in) throws IOException, InvalidDeltaException {
        System.out.println("Applying delta to " + this.toString());
        // For each field, read whether there is a change
        if (in.readBoolean()) {
            // Read the change and apply it to the object
            this.intVal = in.readInt();
            System.out.println(" Applied delta to field 'intVal' = "
                        + this.intVal);
        }
        if (in.readBoolean()) {
            this.doubleVal = in.readDouble();
            System.out.println(" Applied delta to field 'doubleVal' = "
                        + this.doubleVal);
        }
    }
    // In the setter methods, add setting of delta-related
    // fields indicating what has changed
    public void setIntVal(int anIntVal) {
        this.intFldChd = true;
        this.intVal = anIntVal;
    }

    public void setDoubleVal(double aDoubleVal) {
        this.dblFldChd = true;
        this.doubleVal = aDoubleVal;
    }

    public String toString() {
        return "SimpleDelta [ hasDelta = " + hasDelta() + ", intVal = " +
                this.intVal + ", doubleVal = {" + this.doubleVal + "} ]";
    }
}
```

Querying

VMware GemFire provides a SQL-like querying language called OQL that allows you to access data stored in VMware GemFire regions.

Since VMware GemFire regions are key-value stores where values can range from simple byte arrays to complex nested objects, VMware GemFire uses a query syntax based on OQL (Object Query Language) to query region data. OQL is very similar to SQL, but OQL allows you to query complex objects, object attributes, and methods.

- **Querying FAQ and Examples**

  This topic answers some frequently asked questions on querying functionality. It provides examples to help you get started with VMware GemFire querying.

- **Querying with OQL**

  This section provides a high-level introduction to VMware GemFire querying such as building a query string and describes query language features.

- **Advanced Querying**

  This section includes advanced querying topics such as using query indexes, using query bind parameters, querying partitioned regions and query debugging.

- **Working with Indexes**

The VMware GemFire query engine supports indexing. An index can provide significant performance gains for query execution.

Querying FAQ and Examples

This topic answers some frequently asked questions on querying functionality. It provides examples to help you get started with VMware GemFire querying.

For additional information on VMware GemFire querying, see Querying.

- How do I write and execute a query against a VMware GemFire region?
- Can I see query string examples, listed by query type?
- Which APIs should I use to write my queries?
- How do I invoke an object's method in a query?
- Can I invoke a static method on an object in a query?
- How do I write a reusable query?
- When should I create indexes to use in my queries?
- How do I create an index?
- Can I query a partitioned region? Can I perform a join query on a partitioned region?
- How can I improve the performance of a partitioned region query?
- Which query language elements are supported in VMware GemFire?
- How do I debug queries?
- Can I use implicit attributes or methods in my query?
- How do I perform a case-insensitive search on a field in OQL?

How do I write and execute a query against a VMware GemFire region?

To write and execute a query in VMware GemFire, you can use any of the following mechanisms. Sample query code follows.

- VMware GemFire querying APIs
- gfsh command-line interface; in particular the query command
- REST API query endpoints

Sample VMware GemFire Query Code (Java)

```
// Identify your query string.
 String queryString = "SELECT * FROM /exampleRegion";

 // Get QueryService from Cache.
 QueryService queryService = cache.getQueryService();

 // Create the Query Object.
 Query query = queryService.newQuery(queryString);
 // Execute Query locally. Returns results set.
 SelectResults results = (SelectResults)query.execute();

 // Find the Size of the ResultSet.
 int size = results.size();

 // Iterate through your ResultSet.
 Portfolio p = (Portfolio)results.iterator().next(); /* Region containing Portfolio ob
ject. */
```

Can I see query string examples, listed by query type?

The following example query strings use the `/exampleRegion` whose keys are the portfolio ID and whose values correspond to the summarized data shown in the following class definitions:

```
class Portfolio implements DataSerializable {
   int ID;
   String type;
   String status;
   Map positions;
}
class Position implements DataSerializable {
   String secId;
   double mktValue;
   double qty;
}
```

Basic WHERE Clause Examples

In the following examples, the status field is type String and the ID field is type int. See Supported Literals for a complete list of literals supported in VMware GemFire querying.

1. Select all active portfolios.

   ```
   SELECT * FROM /exampleRegion WHERE status = 'active'
   ```

2. Select all portfolios whose status begins with ‘activ’.

   ```
   SELECT * FROM /exampleRegion p WHERE p.status LIKE 'activ%'
   ```

3. Select all portfolios whose ID is greater than 100.

   ```
   SELECT * from /exampleRegion p WHERE p.ID > 100
   ```

Using DISTINCT

Select distinct Objects from the region that satisfy the where clause condition of status = ‘active’.

```
SELECT DISTINCT * FROM /exampleRegion WHERE status = 'active'
```

Aliases and Synonyms

In the query string, the path expressions (region and its objects) can be defined using an alias. This alias can be used or referred to in other places in the query.

```
SELECT DISTINCT * FROM /exampleRegion p WHERE p.status = 'active'
```

```
SELECT p.ID, p.status FROM /exampleRegion p WHERE p.ID > 0
```

Using the NOT Operator

See Operators for a complete list of supported operators.

```
SELECT DISTINCT * FROM /exampleRegion WHERE NOT (status = 'active') AND ID = 2
```

```
SELECT * FROM /exampleRegion WHERE NOT (ID IN SET(1,2))
```

Using the AND and OR Operators

See Operators for a complete list of supported operators.

```
SELECT * FROM /exampleRegion WHERE ID > 4 AND ID < 9
```

```
SELECT * FROM /exampleRegion WHERE ID = 0 OR ID = 1
```

```
SELECT DISTINCT p.status FROM /exampleRegion p
WHERE (p.createTime IN SET (10|) OR p.status IN SET ('active')) AND p.ID > 0
```

Using not equal to

```
SELECT * FROM /exampleRegion portfolio WHERE portfolio.ID <> 2
```

```
SELECT * FROM /exampleRegion portfolio WHERE portfolio.ID != 2
```

Projection attribute example

```
SELECT p.get('account') FROM /exampleRegion p
```

Querying nested collections

The following query uses Positions of type HashMap.

```
SELECT p, pos FROM /exampleRegion p, p.positions.values pos WHERE pos.secId = 'VMW'
```

Using LIMIT

```
SELECT * FROM /exampleRegion p WHERE p.ID > 0 LIMIT 2
```

Using MIN and MAX

See MIN and MAXfor more information.

```
SELECT MIN(ID)
FROM /exampleRegion
WHERE ID > 0
```

```
SELECT MAX(ID)
FROM /exampleRegion
WHERE ID > 0 AND status LIKE 'act%'
```

```
SELECT MIN(pos.mktValue)
FROM /exampleRegion p, p.positions.values pos
WHERE p.ID = 10
```

```
SELECT MAX(p.ID)
FROM /exampleRegion p, p.positions.values pos
WHERE p.ID > 0 OR p.status = 'active' OR pos.secId = 'IBM'
```

Using AVG

See AVG for more information.

```
SELECT AVG(ID)
FROM /exampleRegion
WHERE ID > 0
```

```
SELECT AVG(ID)
FROM /exampleRegion
WHERE ID > 0 AND status LIKE 'act%'
```

```
SELECT AVG(pos.mktValue)
FROM /exampleRegion p, p.positions.values pos
WHERE p.isActive()
```

```
SELECT AVG(DISTINCT p.ID)
FROM /exampleRegion p, p.positions.values pos
WHERE p.ID > 0 OR p.status = 'active' OR pos.secId = 'IBM'
```

Using COUNT

See COUNT for more information.

```
SELECT COUNT(*)
FROM /exampleRegion
WHERE ID > 0
```

```
SELECT COUNT(*)
FROM /exampleRegion
WHERE ID > 0 LIMIT 50
```

```
SELECT COUNT(*)
FROM /exampleRegion
WHERE ID > 0 AND status LIKE 'act%'
```

```
SELECT COUNT(*)
FROM /exampleRegion
WHERE ID IN SET(1,2,3,4,5)
```

```
SELECT COUNT(DISTINCT p.status)
FROM /exampleRegion p
WHERE p.ID > 0
```

```
SELECT COUNT(*)
FROM /exampleRegion p, p.positions.values pos
WHERE p.ID > 0 AND pos.secId 'IBM'
```

```
SELECT DISTINCT COUNT(*)
FROM /exampleRegion p, p.positions.values pos
WHERE p.ID > 0 OR p.status = 'active' OR pos.secId = 'IBM'
```

Using SUM

See SUM for more information.

```
SELECT SUM(ID)
FROM /exampleRegion
WHERE ID > 0
```

```
SELECT SUM(ID)
FROM /exampleRegion
WHERE ID > 0 AND status LIKE 'act%'
```

```
SELECT SUM(pos.mktValue)
FROM /exampleRegion p, p.positions.values pos
WHERE p.status = 'active'
```

```
SELECT SUM(DISTINCT p.ID)
FROM /exampleRegion p, p.positions.values pos
WHERE p.ID > 0 OR p.status = 'active' OR pos.secId = 'IBM'
```

Using GROUP BY

See GROUP BY for more information.

```
SELECT p.status, MAX(p.ID)
FROM /exampleRegion p
WHERE p.ID > 0
GROUP BY p.status
```

```
SELECT p.ID, MIN(pos.qty) AS lessQty
FROM /exampleRegion p, p.positions.values pos
WHERE p.ID > 0 AND p.status = 'active'
GROUP BY p.ID
ORDER BY lessQty ASC
```

```
SELECT p.ID, MAX(pos.mktValue) AS maxValue
FROM /exampleRegion p, p.positions.values pos
WHERE p.ID > 0 AND p.status = 'active'
GROUP BY p.ID
ORDER BY maxValue DESC
```

```
SELECT p.status, AVG(p.ID)
FROM /exampleRegion p
WHERE p.ID > 0
GROUP BY p.status
```

```
SELECT p.ID, pos.secId, AVG(pos.mktValue)
FROM /exampleRegion p, p.positions.values pos
WHERE p.ID > 0 OR p.status = 'active'
GROUP BY p.ID, pos.secId
```

```
SELECT p.status, AVG(p.ID) as sm
FROM /exampleRegion p
WHERE p.ID > 0
GROUP BY p.status
ORDER BY sm DESC
```

```
SELECT p.status, COUNT(*)
FROM /exampleRegion p
WHERE p.ID > 0
GROUP BY p.status
```

```
SELECT p.ID, COUNT(pos) AS positionsAmount
FROM /exampleRegion p, p.positions.values pos
WHERE p.ID > 0 OR p.status = 'active'
GROUP BY p.ID
ORDER BY positionsAmount
```

```
SELECT p.status, SUM(p.ID)
FROM /exampleRegion p
WHERE p.ID > 0
GROUP BY p.status
```

```
SELECT p.ID, pos.secId, SUM(pos.mktValue)
FROM /exampleRegion p, p.positions.values pos
WHERE p.ID > 0 OR p.status = 'active'
GROUP BY p.ID, pos.secId
```

```
SELECT p.status, SUM(p.ID) as sm
FROM /exampleRegion p
WHERE p.ID > 0
GROUP BY p.status
ORDER BY sm DESC
```

```
SELECT p.ID, SUM(pos.mktValue) AS marketValue
FROM /exampleRegion p, p.positions.values pos
WHERE p.isActive()
GROUP BY p.ID
ORDER BY marketValue DESC
```

Using LIKE

```
SELECT * FROM /exampleRegion ps WHERE ps.pkid LIKE '_bc'
```

```
SELECT * FROM /exampleRegion ps WHERE ps.status LIKE '_b_' OR ps.pkid = '2'
```

```
SELECT * FROM /exampleRegion ps WHERE ps.status LIKE '%b%
```

Using Region Entry Keys and Values

```
SELECT * FROM /exampleRegion.keys k WHERE k.ID = 1
```

```
SELECT entry.value FROM /exampleRegion.entries entry WHERE entry.key = '1'
```

```
SELECT key, positions FROM /exampleRegion.entrySet, value.positions.values positions
WHERE positions.mktValue >= 25.00
```

```
SELECT DISTINCT entry.value FROM /exampleRegion.entries entry WHERE entry.key = '1'
```

```
SELECT * FROM /exampleRegion.entries entry WHERE entry.value.ID > 1
```

```
SELECT * FROM /exampleRegion.keySet key WHERE key = '1'
```

```
SELECT * FROM /exampleRegion.values portfolio
WHERE portfolio.status = 'active'
```

Nested Queries

```
IMPORT "query".Portfolio;
SELECT * FROM /exampleRegion, (SELECT DISTINCT * FROM /exampleRegion p TYPE Portfolio,
p.positions
WHERE value!=null)
```

```
SELECT DISTINCT * FROM (SELECT DISTINCT * FROM /exampleRegion portfolios, positions po
s)
WHERE pos.value.secId = 'IBM'
```

```
SELECT * FROM /exampleRegion portfolio
WHERE portfolio.ID IN (SELECT p2.ID FROM /exampleRegion2 p2 WHERE p2.ID > 1)
```

```
SELECT DISTINCT * FROM /exampleRegion p, (SELECT DISTINCT pos
FROM /exampleRegion x, x.positions.values pos WHERE x.ID = p.ID ) AS itrX
```

Query the results of a FROM clause expression

```
SELECT DISTINCT * FROM (SELECT DISTINCT * FROM /Portfolios ptf, positions pos) p
WHERE p.get('pos').value.secId = 'IBM'
```

Hash Map Query

Query using a hashmap. In the following examples, ‘version’ is one of the keys in the hashmap.

```
SELECT * FROM /exampleRegion p WHERE p['version'] = '1.0'
```

```
SELECT entry.key, entry.value FROM /exampleRegion.entries entry
WHERE entry.value['version'] = '100'
```

Map example where “map” is a nested HashMap object

```
SELECT DISTINCT * FROM /exampleRegion p WHERE p.portfolios['key2'] >= 3
```

Example Queries that Fetch Array Values

```
SELECT * FROM /exampleRegion p WHERE p.names[0] = 'aaa'
```

```
SELECT * FROM /exampleRegion p WHERE p.collectionHolderMap.get('1').arr[0] = '0'
```

Using ORDER BY (and ORDER BY with LIMIT)

You must use the DISTINCT keyword with ORDER BY queries.

```
SELECT DISTINCT * FROM /exampleRegion WHERE ID < 101 ORDER BY ID
```

```
SELECT DISTINCT * FROM /exampleRegion WHERE ID < 101 ORDER BY ID asc
```

```
SELECT DISTINCT * FROM /exampleRegion WHERE ID < 101 ORDER BY ID desc
```

```
SELECT DISTINCT key.ID, key.status AS st FROM /exampleRegion.keys key
WHERE key.status = 'inactive' ORDER BY key.status desc, key.ID LIMIT 1
```

```
SELECT DISTINCT * FROM /exampleRegion p ORDER BY p.getP1().secId, p.ID dec, p.ID LIMIT
9
```

```
SELECT DISTINCT * FROM /exampleRegion p ORDER BY p.ID, val.secId LIMIT 1
```

```
SELECT DISTINCT e.key FROM /exampleRegion.entrySet e ORDER BY e.key.ID desc, e.key.pki
d desc
```

```
SELECT DISTINCT p.names[1] FROM /exampleRegion p ORDER BY p.names[1]
```

Join Queries

```
SELECT * FROM /exampleRegion portfolio1, /exampleRegion2 portfolio2
WHERE portfolio1.status = portfolio2.status
```

```
SELECT portfolio1.ID, portfolio2.status FROM /exampleRegion portfolio1, /exampleRegion
2 portfolio2
WHERE portfolio1.status = portfolio2.status
```

```
SELECT * FROM /exampleRegion portfolio1, portfolio1.positions.values positions1,
/exampleRegion2 portfolio2, portfolio2.positions.values positions2 WHERE positions1.se
cId = positions1.secId
```

```
SELECT * FROM /exampleRegion portfolio1, portfolio1.positions.values positions1,
/exampleRegion2 portfolio2, portfolio2.positions.values positions2 WHERE portfolio1.ID
= 1
AND positions1.secId = positions1.secId
```

```
SELECT DISTINCT a, b.price FROM /exampleRegoin1 a, /exampleRegion2 b WHERE a.price =
b.price
```

Using AS

```
SELECT * FROM /exampleRegion p, p.positions.values AS pos WHERE pos.secId != '1'
```

Using TRUE

```
SELECT DISTINCT * FROM /Portfolios WHERE TRUE
```

Using IN and SET

See also IN and SET.

```
SELECT * FROM /exampleRegion portfolio WHERE portfolio.ID IN SET(1, 2)
```

```
SELECT * FROM /exampleRegion portfolio, portfolio.positions.values positions
WHERE portfolio.Pk IN SET ('1', '2') AND positions.secId = '1'
```

```
SELECT * FROM /exampleRegion portfolio, portfolio.positions.values positions
WHERE portfolio.Pk IN SET ('1', '2') OR positions.secId IN SET ('1', '2', '3')
```

```
SELECT * FROM /exampleRegion portfolio, portfolio.positions.values positions
WHERE portfolio.Pk IN SET ('1', '2') OR positions.secId IN SET ('1', '2', '3')
AND portfolio.status = 'active'
```

Querying for Set values

In the following query, sp is of type Set.

```
SELECT * FROM /exampleRegion WHERE sp = set('20', '21', '22')
```

If the Set (sp) only contains ‘20’ and ‘21’, then the query will evaluate to false. The query compares the two sets and looks for the presence of elements in both sets.

For other collection types like list (sp is of type List), the query can be written as follows:

```
SELECT * FROM /exampleRegion WHERE sp.containsAll(set('20', '21', '22'))
```

Invoking Methods on Objects

See Method Invocations for more information.

```
SELECT * FROM /exampleRegion p WHERE p.length > 1
```

```
SELECT DISTINCT * FROM /exampleRegion p WHERE p.positions.size >= 2
```

```
SELECT DISTINCT * FROM /exampleRegion p WHERE p.positions.isEmpty
```

```
SELECT DISTINCT * FROM /exampleRegion p WHERE p.name.startsWith('Bo')
```

Using Query-Level Debugging

To set debugging on the query level, add the **<trace>** keyword before the query. (If you are using an IMPORT statement, include it before the IMPORT).

```
<trace>
SELECT * from /exampleRegion, positions.values TYPE myclass
```

Using Reserved Words in Queries

To access any method, attribute, or named object that has the same name as a query language reserved word, enclose the name within double quotation marks.

```
SELECT * FROM /exampleRegion WHERE status = 'active' AND "type" = 'XYZ'
```

```
SELECT DISTINCT "type" FROM /exampleRegion WHERE status = 'active'
```

Using IMPORT

In the case where the same class name resides in two different namescopes (packages), there needs to be a means of referring to different classes of the same name. The IMPORT statement is used to establish a namescope for a class in a query.

```
IMPORT package.Position;
SELECT DISTINCT * FROM /exampleRegion, positions.values positions TYPE Position WHERE
positions.mktValue >= 25.00
```

Using TYPE

Specifying object type helps the query engine to process the query at optimal speed. Apart from specifying the object types during configuration (using key-constraint and value-constraint), type can be explicitly specified in the query string.

```
SELECT DISTINCT * FROM /exampleRegion, positions.values positions TYPE Position WHERE
positions.mktValue >= 25.00
```

Using ELEMENT

Using ELEMENT(expr) extracts a single element from a collection or array. This function throws a `FunctionDomainException` if the argument is not a collection or array with exactly one element.

```
ELEMENT(SELECT DISTINCT * FROM /exampleRegion WHERE id = 'XYZ-1').status = 'active'
```

Which APIs should I use to write my queries?

If you are querying a Java application's local cache or querying other members, use org.apache.geode.cache.Cache.getQueryService.

If you are writing a Java client to server query, use org.apache.geode.cache.client.Pool.getQueryService.

How do I invoke an object's method in a query?

To use a method in a query, use the attribute name that maps to the public method you want to invoke. For example:

```
/*valid method invocation*/
SELECT DISTINCT * FROM /exampleRegion p WHERE p.positions.size >= 2 - maps to position
s.size()
```

Can I invoke a static method on an object in a query?

No, you cannot invoke a static method on an object. For example, the following query is invalid.

```
/*invalid method invocation*/
SELECT DISTINCT * FROM /exampleRegion WHERE aDay = Day.Wednesday
```

To work around this limitation, write a reusable query that uses a query bind parameter to invoke the static method. Then at query run time, set the parameter to the static method invocation (`Day.Wednesday`). For example:

```
SELECT DISTINCT * FROM /exampleRegion WHERE aDay = $1
```

How do I write a reusable query?

Using query APIs, you can set query bind parameters that are passed values at query run time. For example:

```
// specify the  query string
 String queryString = "SELECT DISTINCT * FROM /exampleRegion p WHERE p.status = $1";

QueryService queryService = cache.getQueryService();
Query query = queryService.newQuery(queryString);

// set a query bind parameter
Object[] params = new Object[1];
params[0] = "active";

// Execute the query locally. It returns the results set.
SelectResults results = (SelectResults) query.execute(params);

// use the results of the query; this example only looks at the size
 int size = results.size();
```

If you use a query bind parameter in place of a region path in your path expression, the parameter value must reference a collection (and not a String such as the name of the region path.)

See Using Query Bind Parameters for more details.

When should I create indexes to use in my queries?

Determine whether your query's performance will benefit from an index. For example, in the following query, an index on pkid can speed up the query.

```
SELECT DISTINCT * FROM /exampleRegion portfolio WHERE portfolio.pkid = '123'
```

How do I create an index?

An index can be created programmatically using APIs or by using xml. Here are two examples:

Sample Code

```
QueryService qs = cache.getQueryService();
 qs.createIndex("myIndex", "status", "/exampleRegion");
 qs.createKeyIndex("myKeyIndex", "id", "exampleRegion");
```

For more information on using this API, see the JavaDocs.

Sample XML

```
<region name="portfolios">
 <region-attributes . . . >
 </region-attributes>
 <index name="myIndex">
 <functional from-clause="/exampleRegion"
     expression="status"/>
 </index>
 <index name="myKeyIndex">
 <primary-key field="id"/>
 </index>
 <entry>
```

For more details on indexes, see Working with Indexes.

Can I create indexes on overflow regions?

You can create indexes on overflow regions, but you are subject to some limitations. For example, the data contained in the index itself cannot be overflowed to disk. See Using Indexes with Overflow Regions for more information.

Can I query a partitioned region? Can I perform a join query on a partitioned region?

You can query partitioned regions, but there are some limitations. You cannot perform join queries on partitioned regions, however you can perform equi-join queries on colocated partitioned regions by executing a function on a local data set.

For a full list of restrictions, see Partitioned Region Query Restrictions.

How can I improve the performance of a partitioned region query?

If you know the data you need to query, you can target particular nodes in your queries (thus reducing the number of servers the query needs to access) by executing the query with the FunctionService. See Querying a Partitioned Region on a Single Node for details. If you are querying data that has been partitioned by a key or specific field, you should first create a key index and then execute the query using the FunctionService with the key or field as a filter. See Optimizing Queries on Data Partitioned by a Key or Field Value.

Which query language elements are supported in VMware GemFire?

Supported elements		
AND	LIMIT	TO_DATE
AS	LIKE	TYPE
COUNT	NOT	WHERE
DISTINCT	NVL	
ELEMENT	OR	
FROM	ORDER BY	
<HINT>	SELECT	
IMPORT	SET	
IN	<TRACE>	
IS_DEFINED	TRUE	
IS_UNDEFINED		

For more information and examples on using each supported keyword, see Supported Keywords.

How do I debug queries?

You can debug a specific query at the query level by adding the <trace> keyword before the query string that you want to debug. Here is an example:

```
<trace> SELECT * FROM /exampleRegion
```

You can also write:

```
<TRACE> SELECT * FROM /exampleRegion
```

When the query is executed, VMware GemFire will log a message in `$GEMFIRE_DIR/system.log` with the following information:

```
[info 2011/08/29 11:24:35.472 PDT CqServer <main> tid=0x1] Query Executed in 9.619656
ms; rowCount = 99;
indexesUsed(0) "select *  from /exampleRegion"
```

If you want to enable debugging for all queries, you can enable query execution logging by setting a System property on the command line during start-up:

```
gfsh>start server --name=server_name --J=-Dgemfire.Query.VERBOSE=true
```

Or you can set the property programmatically:

```
System.setProperty("gemfire.Query.VERBOSE","true");
```

Can I use implicit attributes or methods in my query?

If an implicit attribute or method name can only be associated with one untyped iterator, the VMware GemFire query processor will assume that it is associated with that iterator. However, if more than one untyped iterator is in scope, then the query will fail with a `TypeMismatchException`. The following query fails because the query processor does not fully type expressions:

```
select distinct value.secId from /pos , getPositions(23)
```

The following query, however, succeeds because the iterator is either explicitly named with a variable or it is typed:

```
select distinct e.value.secId from /pos , getPositions(23) e
```

Can I instruct the query engine to use specific indexes with my queries?

Using HINT *indexname* allows you to instruct the query engine to prefer and filter results from the specified indexes. If you provide multiple index names, the query engine will use all available indexes but prefer the specified indexes.

```
<HINT 'IDIndex'> SELECT * FROM /Portfolios p WHERE p.ID > 10 AND p.owner = 'XYZ'
```

```
<HINT 'IDIndex', 'OwnerIndex'> SELECT * FROM /Portfolios p WHERE p.ID > 10 AND p.owner
= 'XYZ' AND p.value < 100
```

How do I perform a case-insensitive search on a field in OQL?

You can use the Java String class methods `toUpperCase` and `toLowerCase` to transform fields where you want to perform a case-insensitive search. For example:

```
SELECT entry.value FROM /exampleRegion.entries entry WHERE entry.value.toUpperCase LIK
E '%BAR%'
```

or

```
SELECT * FROM /exampleRegion WHERE foo.toLowerCase LIKE '%bar%'
```

Querying with OQL

This section provides a high-level introduction to VMware GemFire querying such as building a query string and describes query language features.

VMware GemFire provides a SQL-like querying language that allows you to access data stored in VMware GemFire regions. Since VMware GemFire regions are key-value stores where values can range from simple byte arrays to complex nested objects, VMware GemFire uses a query syntax based on OQL (Object Query Language) to query region data. OQL and SQL have many syntactical similarities, however they have significant differences. For example, while OQL does not offer all of the capabilities of SQL like aggregates, OQL does allow you to execute queries on complex object graphs, query object attributes and invoke object methods.

The syntax of a typical VMware GemFire OQL query is:

```
[IMPORT package]
SELECT [DISTINCT] projectionList
FROM collection1, [collection2, …]
[WHERE clause]
[ORDER BY order_criteria [desc]]
```

Therefore, a simple VMware GemFire OQL query resembles the following:

```
SELECT DISTINCT * FROM /exampleRegion WHERE status = 'active'
```

An important characteristic of VMware GemFire querying to note is that by default, VMware GemFire queries on the values of a region and not on keys. To obtain keys from a region, you must use the keySet path expression on the queried region. For example, `/exampleRegion.keySet`.

For those new to the VMware GemFire querying, see also the VMware GemFire Querying FAQ and Examples.

Advantages of OQL

The following list describes some of the advantages of using an OQL-based querying language:

- You can query on any arbitrary object
- You can navigate object collections
- You can invoke methods and access the behavior of objects
- Data mapping is supported
- You are not required to declare types. Since you do not need type definitions, you can work across multiple languages

- You are not constrained by a schema

Writing and Executing a Query in VMware GemFire

The VMware GemFire QueryService provides methods to create the Query object. You can then use the Query object to perform query-related operations.

The QueryService instance you should use depends on whether you are querying the local cache of an application or if you want your application to query the server cache.

Querying a Local Cache

To query the application's local cache or to query other members, use `org.apache.geode.cache.Cache.getQueryService`.

Sample Code

```
 // Identify your query string.
 String queryString = "SELECT DISTINCT * FROM /exampleRegion";

 // Get QueryService from Cache.
 QueryService queryService = cache.getQueryService();

 // Create the Query Object.
 Query query = queryService.newQuery(queryString);

 // Execute Query locally. Returns results set.
 SelectResults results = (SelectResults)query.execute();

 // Find the Size of the ResultSet.
 int size = results.size();

 // Iterate through your ResultSet.
 Portfolio p = (Portfolio)results.iterator().next(); /* Region containing Portfolio ob
ject. */
```

Querying a Server Cache from a Client

To perform a client to server query, use `org.apache.geode.cache.client.Pool.getQueryService`.

Sample Code

```
// Identify your query string.
 String queryString = "SELECT DISTINCT * FROM /exampleRegion";

 // Get QueryService from client pool.
 QueryService queryService = pool.getQueryService();

 // Create the Query Object.
 Query query = queryService.newQuery(queryString);

 // Execute Query locally. Returns results set.
 SelectResults results = (SelectResults)query.execute();

 // Find the Size of the ResultSet.
 int size = results.size();

 // Iterate through your ResultSet.
 Portfolio p = (Portfolio)results.iterator().next(); /* Region containing Portfolio ob
ject. */
```

Refer to the following JavaDocs for specific APIs:

- Query package
- QueryService

Note: You can also perform queries using the gfsh `query` command. See query.

Building a Query String

A query string is a fully formed OQL statement that can be passed to a query engine and executed against a data set. To build a query string, you combine supported keywords, expressions, and operators to create an expression that returns the information you require.

A query string follows the rules specified by the query language and grammar. It can include:

- **Namescopes**. For example, the IMPORT statement. See IMPORT Statement.
- **Path expressions**. For example, in the query `SELECT * FROM /exampleRegion`,`/exampleRegion` is a path expression. See FROM Clause.
- **Attribute names**. For example, in the query `SELECT DISTINCT * FROM /exampleRegion p WHERE p.position1.secId = '1'`, we access the `secId` attribute of the Position object. See WHERE Clause.
- **Method invocations**. For example, in the query `SELECT DISTINCT * FROM /exampleRegion p WHERE p.name.startsWith('Bo')`, we invoke the `startsWith` method on the Name object. See WHERE Clause.
- **Operators**. For example, comparison operators (=,<,>,<>), unary operators (NOT), logical operators (AND, OR) and so on. See Operators for a complete list.
- **Literals**. For example, boolean, date, time and so on. See Supported Literals for a complete list.
- **Query bind parameters**. For example, in the query `SELECT DISTINCT * FROM $1 p WHERE p.status = $2`, $1 and $2 are parameters that can be passed to the query during runtime. See Using Query Bind Parameters for more details.
- **Preset query functions**. For example, ELEMENT(expr) and IS_DEFINED(expr). See SELECT Statement for other available functions.
- **SELECT statements**. For example, in the example queries above `SELECT *` or `SELECT DISTINCT *`. See SELECT Statement for other available functions.
- **Comments**. OQL permits extra characters to accompany the query string without changing the string's definition. Form a multi-line comment by enclosing the comment body within `/*` and `*/` delimiters; OQL does not permit nested comments. A single line comment body is all the characters to the right of `--` (two hyphens) up to the end of the line.

The components listed above can all be part of the query string, but none of the components are required. At a minimum, a query string contains an expression that can be evaluated against specified data.

The following sections provide guidelines for the query language building blocks that are used when writing typical VMware GemFire queries.

- IMPORT Statement
- FROM Clause
- WHERE Clause
- SELECT Statement
- OQL Aggregate Functions

IMPORT Statement

It is sometimes necessary for an OQL query to refer to the class of an object. In cases where the same class name resides in two different namescopes (packages), you must be able to differentiate the classes having the same name.

The **IMPORT** statement is used to establish a name for a class in a query.

```
IMPORT package.Position;
SELECT DISTINCT * FROM /exampleRegion, positions.values positions TYPE Position WHERE
positions.mktValue >= 25.00
```

FROM Clause

Use the FROM clause to bring the data you need into scope for the rest of your query. The FROM clause also includes object typing and iterator variables.

The query engine resolves names and path expressions according to the name space that is currently in scope in the query.

Path Expressions

The initial name space for any query is composed of:

- **Regions.** In the context of a query, the name of a region is specified by its full path starting with a forward slash (/) and delimited by the forward slash between region names. For example, **/exampleRegion** or **/root/exampleRegion**.
- **Region querying attributes**. From a region path, you can access the Region object's public fields and methods, referred to in querying as the region's attributes. For example, **/exampleRegion.size**.
- **Top-level region data.** You can access entry keys and entry data through the region path.
    1. `/exampleRegion.keySet` returns the Set of entry keys in the region
    2. `/exampleRegion.entryset` returns the Set of Region.Entry objects
    3. `/exampleRegion.values` returns the Collection of entry values
    4. `/exampleRegion` returns the Collection of entry values

New name spaces are brought into scope based on the FROM clause in the SELECT statement.

Examples:

Query a region for all distinct values. Return a collection of unique entry values from the region:

```
SELECT DISTINCT * FROM /exampleRegion
```

Query the top level region data using entrySet. Return the keys and positions of Region.Entry objects whose mktValue attribute is greater than 25.00:

```
SELECT key, positions FROM /exampleRegion.entrySet, value.positions.values positions W
HERE positions.mktValue >= 25.00
```

Query the region for its entry values. Return a set of unique values from Region.Entry objects that have the key equal to 1:

```
SELECT DISTINCT entry.value FROM /exampleRegion.entries entry WHERE entry.key = '1'
```

Query the region for its entry values. Return the set of all entry values in which the `ID` field is greater than 1000:

```
SELECT * FROM /exampleRegion.entries entry WHERE entry.value.ID > 1000
```

Query entry keys in the region. Return a set of entry keys in the region that have the key equal to '1':

```
SELECT * FROM /exampleRegion.keySet key WHERE key = '1'
```

Query values in the region. Return a collection of entry values in the region that have the status attribute value of 'active':

```
SELECT * FROM /exampleRegion.values portfolio WHERE portfolio.status = 'active'
```

Aliases and Synonyms

In query strings, you can use aliases in path expressions (region and its objects) so that you can refer to the region or objects in other places in the query.

You can also use the **AS** keyword to provide a label for joined path expressions.

Examples:

```
SELECT DISTINCT * FROM /exampleRegion p WHERE p.status = 'active'
```

```
SELECT * FROM /exampleRegion p, p.positions.values AS pos WHERE pos.secId != '1'
```

Object Typing

Specifying object type in the FROM clause helps the query engine to process the query at optimal speed. Apart from specifying the object types during configuration (using key-constraint and value-constraint), type can be explicitly specified in the query string.

Example:

```
SELECT DISTINCT * FROM /exampleRegion, positions.values positions TYPE Position WHERE
positions.mktValue >= 25.00
```

WHERE Clause

Each FROM clause expression must resolve to a collection of objects. The collection is then available for iteration in the query expressions that follow in the WHERE clause.

For example:

```
SELECT DISTINCT * FROM /exampleRegion p WHERE p.status = 'active'
```

The entry value collection is iterated by the WHERE clause, comparing the status field to the string ‘active’. When a match is found, the value object of the entry is added to the return set.

In the next example query, the collection specified in the first FROM clause expression is used by the rest of the SELECT statement, including the second FROM clause expression.

```
SELECT DISTINCT * FROM /exampleRegion, positions.values p WHERE p.qty > 1000.00
```

Implementing equals and hashCode Methods

You must implement the `equals` and `hashCode` methods in your custom objects if you are doing ORDER BY and DISTINCT queries on the objects. The methods must conform to the properties and behavior documented in the online Java API documentation for `java.lang.Object`. Inconsistent query results may occur if these methods are absent.

If you have implemented `equals` and `hashCode` methods in your custom objects, you must provide detailed implementations of these methods so that queries execute properly against the objects. For example, assume that you have defined a custom object (CustomObject) with the following variables:

```
int ID
int otherValue
```

Let’s put two CustomObjects (we’ll call them CustomObjectA and CustomObjectB) into the cache:

CustomObjectA:

```
ID=1
otherValue=1
```

CustomObjectB:

```
ID=1
otherValue=2
```

If you have implemented the equals method to simply match on the ID field (ID == ID), queries will produce unpredictable results.

The following query:

```
SELECT * FROM /CustomObjects c
WHERE c.ID > 1 AND c.ID < 3
AND c.otherValue > 0 AND c.otherValue < 3
```

returns two objects, however the objects will be two of either CustomObjectA or CustomObjectB.

Alternately, the following query:

```
SELECT * FROM /CustomObjects c
WHERE c.ID > 1 AND c.ID < 3
AND c.otherValue > 1 AND c.otherValue < 3
```

returns either 0 results or 2 results of CustomObjectB, depending on which entry is evaluated last.

To avoid unpredictable querying behavior, implement detailed versions of the `equals` and `hashCode` methods.

If you are comparing a non-primitive field of the object in the WHERE clause, use the `equals` method instead of the `=` operator. For example instead of `nonPrimitiveObj = objToBeCompared` use `nonPrimitiveObj.equals(objToBeCompared)`.

Querying Serialized Objects

Objects must implement serializable if you will be querying partitioned regions or if you are performing client-server querying.

If you are using PDX serialization, you can access the values of individual fields without having to deserialize the entire object. This is accomplished by using PdxInstance, which is a wrapper around the serialized stream. The PdxInstance provides a helper method that takes field-name and returns the value without deserializing the object. While evaluating the query, the query engine will access field values by calling the getField method thus avoiding deserialization.

To use PdxInstances in querying, ensure that PDX serialization reads are enabled in your server's cache. In gfsh, execute the following command before starting up your data members:

```
gfsh>configure pdx --read-serialized=true
```

See configure pdx for more information.

In cache.xml, set the following:

```
// Cache configuration setting PDX read behavior
<cache>
  <pdx read-serialized="true">
  ...
  </pdx>
</cache>
```

Attribute Visibility

You can access any object or object attribute that is available in the current scope of a query. In querying, an object's attribute is any identifier that can be mapped to a public field or method in the object. In the FROM specification, any object that is in scope is valid. Therefore, at the beginning of a query, all locally cached regions and their attributes are in scope.

For attribute Position.secId which is public and has getter method "getSecId()", the query can be written as the following:

```
SELECT DISTINCT * FROM /exampleRegion p WHERE p.position1.secId = '1'
SELECT DISTINCT * FROM /exampleRegion p WHERE p.position1.SecId = '1'
SELECT DISTINCT * FROM /exampleRegion p WHERE p.position1.getSecId() = '1'
```

The query engine tries to evaluate the value using the public field value. If a public field value is not found, it makes a get call using field name (note that the first character is uppercase.)

Joins

If collections in the FROM clause are not related to each other, the WHERE clause can be used to join them.

The statement below returns all portfolios from the /exampleRegion and /exampleRegion2 regions that have the same status.

```
SELECT * FROM /exampleRegion portfolio1, /exampleRegion2 portfolio2 WHERE portfolio1.s
tatus = portfolio2.status
```

To create indexes for region joins you create single-region indexes for both sides of the join condition. These are used during query execution for the join condition. Partitioned regions do not support region joins. For more information on indexes, see Working with Indexes.

Examples:

Query two regions. Return the ID and status for portfolios that have the same status.

```
SELECT portfolio1.ID, portfolio2.status FROM /exampleRegion portfolio1, /exampleRegion
2 portfolio2 WHERE portfolio1.status = portfolio2.status
```

Query two regions, iterating over all `positions` within each portfolio. Return all 4-tuples consisting of the value from each of the two regions and the value portion of the `positions` map from both regions in which the `secId` field of positions match.

```
SELECT * FROM /exampleRegion portfolio1, portfolio1.positions.values positions1, /exam
pleRegion2 portfolio2, portfolio2.positions.values positions2 WHERE positions1.secId =
positions2.secId
```

Same query as the previous example, with the additional constraint that matches will have a `ID` of 1.

```
SELECT * FROM /exampleRegion portfolio1, portfolio1.positions.values positions1, /exam
pleRegion2 portfolio2, portfolio2.positions.values positions2 WHERE portfolio1.ID = 1
AND positions1.secId = positions2.secId
```

LIKE

VMware GemFire offers limited support for the LIKE predicate. LIKE can be used to mean ‘equals to’. If you terminate the string with a wildcard (‘%’), it behaves like ‘starts with’. You can also place a wildcard (either ‘%’ or ‘_’) at any other position in the comparison string. You can escape the wildcard characters to represent the characters themselves.

Note: The ‘*’ wildcard is not supported in OQL LIKE predicates.

You can also use the LIKE predicate when an index is present.

Examples:

Query the region. Return all objects where status equals ‘active’:

```
SELECT * FROM /exampleRegion p WHERE p.status LIKE 'active'
```

Query the region using a wild card for comparison. Returns all objects where status begins with ‘activ’:

```
SELECT * FROM /exampleRegion p WHERE p.status LIKE 'activ%'
```

Case Insensitive Fields

You can use the Java String class methods `toUpperCase` and `toLowerCase` to transform fields where you want to perform a case-insensitive search. For example:

```
SELECT entry.value FROM /exampleRegion.entries entry WHERE entry.value.toUpperCase LIK
E '%BAR%'
```

or

```
SELECT * FROM /exampleRegion WHERE foo.toLowerCase LIKE '%bar%'
```

Method Invocations

To use a method in a query, use the attribute name that maps to the public method you want to invoke, or directly use the public method name instead. It is important to note that when you use the attribute name instead of the method name, VMware GemFire will search for public methods named as the attribute itself or public methods with the `get` prefix.

```
SELECT r.id FROM /exampleRegion r                                  - maps to obje
ct.id() or object.getId()
SELECT q.getName() FROM /exampleRegion q                           - maps to obje
ct.getName()
SELECT DISTINCT * FROM /exampleRegion p WHERE p.positions.size >= 2     - maps to posi
tions.size()
```

Methods declared to return void evaluate to `null` when invoked through the query processor.

You cannot invoke a static method. See Enum Objects for more information.

Methods without parameters

If the attribute name maps to a public method that takes no parameters, just include the method name in the query string as an attribute. For example, `emps.isEmpty` is equivalent to `emps.isEmpty()`.

In the following example, the query invokes `isEmpty` on positions, and returns the set of all portfolios with no positions:

```
SELECT DISTINCT * FROM /exampleRegion p WHERE p.positions.isEmpty
```

Methods with parameters

To invoke methods with parameters, include the method name in the query string as an attribute and provide method arguments between parentheses.

This example passes the argument `"Bo"` to the public method, and returns all names that begin with `"Bo"`.

```
SELECT DISTINCT * FROM /exampleRegion p WHERE p.name.startsWith('Bo')
```

For overloaded methods, the query processor decides which method to call by matching the runtime argument types with the parameter types required by the method. If only one method's signature matches the parameters provided, it is invoked. The query processor uses runtime types to match method signatures.

If more than one method can be invoked, the query processor chooses the method whose parameter types are the most specific for the given arguments. For example, if an overloaded method includes versions with the same number of arguments, but one takes a `Person` type as an argument and the other takes an `Employee` type, derived from `Person`, `Employee` is the more specific object type. If the argument passed to the method is compatible with both types, the query processor uses the method with the `Employee` parameter type.

The query processor uses the runtime types of the parameters and the receiver to determine the proper method to invoke. Because runtime types are used, an argument with a `null` value has no typing information, and so can be matched with any object type parameter. When a `null` argument is used, if the query processor cannot determine the proper method to invoke based on the non-null arguments, it throws an `AmbiguousNameException`.

Methods calls with the `SecurityManager` enabled

When the `SecurityManager` is enabled, by default VMware GemFire throws a `NotAuthorizedException` when any method that does not belong to the to the list of default allowed methods, given in RestrictedMethodAuthorizer, is invoked.

In order to further customize this authorization check, see Changing the Method Authorizer.

In the past you could use the system property `gemfire.QueryService.allowUntrustedMethodInvocation` to disable the check altogether, but this approach is deprecated and will be removed in future releases; you need to configure the UnrestrictedMethodAuthorizer instead.

Enum Objects

To write a query based on the value of an Enum object field, you must use the `toString` method of the enum object or use a query bind parameter.

For example, the following query is NOT valid:

```
//INVALID QUERY
select distinct * from /QueryRegion0 where aDay = Day.Wednesday
```

The reason it is invalid is that the call to `Day.Wednesday` involves a static class and method invocation which is not supported.

Enum types can be queried by using toString method of the enum object or by using bind parameter. When you query using the toString method, you must already know the constraint value that you wish to query. In the following first example, the known value is ‘active’.

Examples:

Query enum type using the toString method:

```
// eStatus is an enum with values 'active' and 'inactive'
select * from /exampleRegion p where p.eStatus.toString() = 'active'
```

Query enum type using a bind parameter. The value of the desired Enum field (Day.Wednesday) is passed as an execution parameter:

```
select distinct * from /QueryRegion0 where aDay = $1
```

IN and SET

The IN expression is a boolean indicating if one expression is present inside a collection of expressions of compatible type. The determination is based on the expressions’ equals semantics.

If `e1` and `e2` are expressions, `e2` is a collection, and `e1` is an object or a literal whose type is a subtype or the same type as the elements of `e2`, then `e1 IN e2` is an expression of type boolean.

The expression returns:

- TRUE if e1 is not UNDEFINED and is contained in collection e2
- FALSE if e1 is not UNDEFINED and is not contained in collection e2 #
- UNDEFINED if e1 is UNDEFINED

For example, `2 IN SET(1, 2, 3)` is TRUE.

Another example is when the collection you are querying into is defined by a subquery. This query looks for companies that have an active portfolio on file:

```
SELECT name, address FROM /company
  WHERE id IN (SELECT id FROM /portfolios WHERE status = 'active')
```

The interior SELECT statement returns a collection of ids for all /portfolios entries whose status is active. The exterior SELECT iterates over /company, comparing each entry’s id with this collection. For each entry, if the IN expression returns TRUE, the associated name and address are added to the outer SELECT’s collection.

Comparing Set Values

The following is an example of a set value type comparison where sp is of type Set:

```
SELECT * FROM /exampleRegion WHERE sp = set('20','21','22')
```

In this case, if sp contains only ‘20’ and ‘21’, then the query evaluates to false. The query compares the two sets and looks for the presence of all elements in both sets.

For other collections types like list, the query can be written as follows:

```
SELECT * FROM /exampleRegion WHERE sp.containsAll(set('20','21','22))
```

where sp is of type List.

In order to use it for Set value, the query can be written as:

```
SELECT * FROM /exampleRegion WHERE sp IN SET (set('20','21','22'),set('10',11','12'))
```

where a set value is searched in collection of set values.

One problem is that you cannot create indexes on Set or List types (collection types) that are not comparable. To workaround this, you can create an index on a custom collection type that implements Comparable.

Double.NaN and Float.NaN Comparisons

The comparison behavior of Double.NaN and Float.NaN within VMware GemFire queries follow the semantics of the JDK methods Float.compareTo and Double.compareTo.

In summary, the comparisons differ in the following ways from those performed by the Java language numerical comparison operators (<, <=, ==, >= >) when applied to primitive double [float] values:

- Double.NaN [Float.NaN] is considered to be equal to itself and greater than all other double [float] values (including Double.POSITIVE_INFINITY [Float.POSITIVE_INFINITY]).
- 0.0d [0.0f] is considered by this method to be greater than -0.0d [-0.0f].

Therefore, Double.NaN[Float.NaN] is considered to be larger than Double.POSITIVE_INFINITY[Float.POSITIVE_INFINITY]. Here are some example queries and what to expect.

If p.value is NaN, the following query:	Evaluates to:	Appears in the result set?
`SELECT * FROM /positions p WHERE p.value = 0`	false	no
`SELECT * FROM /positions p WHERE p.value > 0`	true	yes
`SELECT * FROM /positions p WHERE p.value >= 0`	true	yes
`SELECT * FROM /positions p WHERE p.value < 0`	false	no
`SELECT * FROM /positions p WHERE p.value <= 0`	false	no
When p.value and p.value1 are both NaN, the following query:	**Evaluates to:**	**Appears in the result set:**
`SELECT * FROM /positions p WHERE p.value = p.value1`	true	yes

If you combine values when defining the following query in your code, when the query is executed the value itself is considered UNDEFINED when parsed and will not be returned in the result set.

```
String query = "SELECT * FROM /positions p WHERE p.value =" + Float.NaN
```

Executing this query, the value itself is considered UNDEFINED when parsed and will not be returned in the result set.

To retrieve NaN values without having another field already stored as NaN, you can define the following query in your code:

```
String query = "SELECT * FROM /positions p WHERE p.value > " + Float.MAX_VALUE;
```

Arithmetic Operations

Arithmetic operators may be used in any expression.

For example, this query selects all people with a body mass index less than 25:

```
String query = "SELECT * FROM /people p WHERE p.height * p.height/p.weight < 25";
```

SELECT Statement

The SELECT statement allows you to filter data from the collection of object(s) returned by a WHERE search operation. The projection list is either specified as * or as a comma delimited list of expressions.

For *, the interim results of the WHERE clause are returned from the query.

Examples:

Query all objects from the region using *. Returns the Collection of portfolios (The exampleRegion contains Portfolio as values).

```
SELECT * FROM /exampleRegion
```

Query secIds from positions. Returns the Collection of secIds from the positions of active portfolios:

```
SELECT secId FROM /exampleRegion, positions.values TYPE Position
WHERE status = 'active'
```

Returns a Collection of struct<type: String, positions: map> for the active portfolios. The second field of the struct is a Map (jav.utils.Map) object, which contains the positions map as the value:

```
SELECT "type", positions FROM /exampleRegion
WHERE status = 'active'
```

Returns a Collection of struct<portfolios: Portfolio, values: Position> for the active portfolios:

```
SELECT * FROM /exampleRegion, positions.values
TYPE Position WHERE status = 'active'
```

Returns a Collection of struct<pflo: Portfolio, posn: Position> for the active portfolios:

```
SELECT * FROM /exampleRegion portfolio, positions positions
TYPE Position WHERE portfolio.status = 'active'
```

SELECT Statement Results

The result of a SELECT statement is either UNDEFINED or is a Collection that implements the SelectResults interface.

The SelectResults returned from the SELECT statement is either:

1. A collection of objects, returned for these two cases:
    - When only one expression is specified by the projection list and that expression is not explicitly specified using the fieldname:expression syntax
    - When the SELECT list is * and a single collection is specified in the FROM clause
2. A collection of Structs that contains the objects

When a struct is returned, the name of each field in the struct is determined following this order of preference:

1. If a field is specified explicitly using the fieldname:expression syntax, the fieldname is used.
2. If the SELECT projection list is * and an explicit iterator expression is used in the FROM clause, the iterator variable name is used as the field name.
3. If the field is associated with a region or attribute path, the last attribute name in the path is used.
4. If names cannot be decided based on these rules, arbitrary unique names are generated by the query processor.

DISTINCT

Use the DISTINCT keyword if you want to limit the results set to unique rows. Note that in the current version of VMware GemFire you are no longer required to use the DISTINCT keyword in your SELECT statement.

```
SELECT DISTINCT * FROM /exampleRegion
```

Note: If you are using DISTINCT queries, you must implement the equals and hashCode methods for the objects that you query.

LIMIT

You can use the LIMIT keyword at the end of the query string to limit the number of values returned.

For example, this query returns at most 10 values:

```
SELECT * FROM /exampleRegion LIMIT 10
```

ORDER BY

You can order your query results in ascending or descending order by using the ORDER BY clause. You must use DISTINCT when you write ORDER BY queries.

```
SELECT DISTINCT * FROM /exampleRegion WHERE ID < 101 ORDER BY ID
```

The following query sorts the results in ascending order:

```
SELECT DISTINCT * FROM /exampleRegion WHERE ID < 101 ORDER BY ID asc
```

The following query sorts the results in descending order:

```
SELECT DISTINCT * FROM /exampleRegion WHERE ID < 101 ORDER BY ID desc
```

Note: If you are using ORDER BY queries, you must implement the equals and hashCode methods for the objects that you query.

Preset Query Functions

VMware GemFire provides several built-in functions for evaluating or filtering data returned from a query. They include the following:

Function	Description	Example
ELEMENT(expr)	Extracts a single element from a collection or array. This function throws a `FunctionDomainException` if the argument is not a collection or array with exactly one element.	`ELEMENT(SELECT DISTINCT * FROM /exampleRegion WHERE id = 'XYZ-1').status = 'active'`
IS_DEFINED(expr)	Returns TRUE if the expression does not evaluate to UNDEFINED. Inequality queries include undefined values in their query results. With the IS_DEFINED function, you can limit results to only those elements with defined values.	`IS_DEFINED(SELECT DISTINCT * FROM /exampleRegion p WHERE p.status = 'active')`
IS_UNDEFINED (expr)	Returns TRUE if the expression evaluates to UNDEFINED. With the exception of inequality queries, most queries do not include undefined values in their query results. The IS_UNDEFINED function allows undefined values to be included, so you can identify elements with undefined values.	`SELECT DISTINCT * FROM /exampleRegion p WHERE IS_UNDEFINED(p.status)`
NVL(expr1, expr2)	Returns expr2 if expr1 is null. The expressions can be query parameters (bind arguments), path expressions, or literals.	

Function	Description	Example
TO_DATE(date_str, format_str)	Returns a Java Data class object. The arguments must be String S with date_str representing the date and format_str representing the format used by date_str. The format_str you provide is parsed using java.text.SimpleDateFormat.	

OQL Aggregate Functions

The aggregate functions `MIN`, `MAX`, `COUNT`, `COUNT` over a `DISTINCT` expression, `SUM`, `SUM` over a `DISTINCT` expression, `AVG`, `AVG` over a `DISTINCT` expression, and `GROUP BY` are supported.

The following sections provide descriptions and limitations (if any) about the aggregate functions, how do they work and how they can be used.

GROUP BY

You can collect data across multiple entries and group the results by one or more columns through the usage of the `GROUP BY` statement. It's important to note some facts about its usage:

- It can group results by one or more fields.
- It returns a single record / entry per group.
- It must always be placed before the `ORDER BY` clause, if any.
- It can be used in conjunction with other aggregate functions: `MIN`, `MAX`, `COUNT`, `SUM` and `AVG`.
- It groups records using the selected fields if and only if the fields have identical data across entries.
- It is required, whenever an aggregate function is used within a query with other selected fields, to also use `GROUP BY`.
- If there are no other aggregate functions within the query, all fields included within a GROUP BY clause must also be part of the original projection list, and all fields included within the projection list must also be part of the `GROUP BY` clause.

The following are example `GROUP BY` queries.

The following `GROUP BY` query returns the maximum amount of sales per employee.

```
SELECT ID, MAX(e.sales)
FROM /employees e
GROUP BY ID
```

The following `GROUP BY` query returns the minimum, maximum, total count, average and summation of IDs grouped by status.

```
SELECT pf.status, MIN(pf.ID), MAX(pf.ID), COUNT(pf.ID), AVG(pf.ID), SUM(pf.ID)
FROM /portfolio pf
GROUP BY pf.status
```

MIN

The `MIN` keyword returns the minimum or smallest value from the selected expression. The expression itself must always evaluate to `java.lang.Comparable`. The `MIN` statement returns the actual type of the selected element as its result.

The following are example `MIN` queries that return region entries (the entries implement the `java.lang.Comparable` interface).

```
SELECT MIN(pf)
FROM /exampleRegion pf
```

```
SELECT MIN(pf)
FROM /exampleRegion
pf WHERE pf.ID > 0
```

```
SELECT MIN(pf)
FROM /exampleRegion pf
WHERE pf.ID > 10 LIMIT 50
```

```
SELECT MIN(pf)
FROM /exampleRegion pf
WHERE pf.ID > 0 AND pf.status LIKE 'act%'
```

The following `MIN` query returns the lowest entry ID that matches the query's selection criteria.

```
SELECT MIN(pf.ID)
FROM /exampleRegion pf, pf.positions.values pos
WHERE pf.ID > 0 AND pos.secId = 'IBM'
```

The following `MIN` query returns the lowest positive ID grouped by status.

```
SELECT pf.status, MIN(pf.ID)
FROM /exampleRegion pf
WHERE pf.ID > 0
GROUP BY pf.status
```

MAX

The `MAX` keyword returns the maximum or highest value from the selected expression. The expression itself must always evaluate to `java.lang.Comparable`. The `MAX` statement returns the type of the selected element as its result.

The following are example `MAX` queries that return region entries (the entries implement the `java.lang.Comparable` interface).

```
SELECT MAX(pf)
FROM /exampleRegion pf
```

```
SELECT MAX(pf)
FROM /exampleRegion
```

```
pf WHERE pf.ID > 0
```

```
SELECT MAX(pf)
FROM /exampleRegion pf
WHERE pf.ID > 10 LIMIT 50
```

```
SELECT MAX(pf)
FROM /exampleRegion pf
WHERE pf.ID > 0 AND pf.status LIKE 'act%'
```

The following `MAX` query returns the highest entry ID that matches the query's selection criteria.

```
SELECT MAX(pf.ID)
FROM /exampleRegion pf, pf.positions.values pos
WHERE pf.ID > 0 AND pos.secId = 'IBM'
```

The following `MAX` query returns the highest positive IDs grouped by status.

```
SELECT pf.status, MAX(pf.ID)
FROM /exampleRegion pf
WHERE pf.ID > 0
GROUP BY pf.status
```

COUNT

The `COUNT` keyword returns the number of results that match the query selection conditions specified in the `WHERE` clause. Using `COUNT` allows you to determine the size of a results set. The `COUNT` statement always returns a `java.lang.Integer` or `java.lang.Long` as the result (depending on how big the value is); you should take this into consideration when executing the query: if an overflow occurs while computing the `COUNT` function because the value is higher than `Long.MAX_VALUE` (2^{63} - 1), the result will be incorrect.

The following queries are example `COUNT` queries that return region entries:

```
SELECT COUNT(*)
FROM /exampleRegion
```

```
SELECT COUNT(*)
FROM /exampleRegion
WHERE ID > 0
```

```
SELECT COUNT(*)
FROM /exampleRegion
WHERE ID > 0 LIMIT 50
```

```
SELECT COUNT(*)
FROM /exampleRegion
WHERE ID > 0 AND status LIKE 'act%'
```

```
SELECT COUNT(*)
FROM /exampleRegion
```

```
WHERE ID IN SET(1,2,3,4,5)
```

The following `COUNT` query returns the total number of StructTypes that match the query's selection criteria.

```
SELECT COUNT(*)
FROM /exampleRegion p, p.positions.values pos
WHERE p.ID > 0 AND pos.secId 'IBM'
```

The following `COUNT` query uses the `DISTINCT` keyword and eliminates duplicates from the number of results.

```
SELECT DISTINCT COUNT(*)
FROM /exampleRegion p, p.positions.values pos
WHERE p.ID > 0 OR p.status = 'active' OR pos.secId = 'IBM'
```

SUM

The `SUM` keyword returns the summation of all results that match the query selection conditions specified in the `WHERE` clause. Using `SUM` allows you to aggregate specific numeric values within a results set. For partitioned regions, each node's buckets compute a sum over that node and return the result to the coordinator node executing the query, which then aggregates the sums across all nodes.

The `SUM` function where the `DISTINCT` modifier is applied to the expression returns the summation over the set of unique (distinct) values. For partitioned regions, the distinct values in a node's buckets are returned to the coordinator node, which can then calculate the sum over the values that are unique across nodes, after eliminating duplicate values that come from separate nodes.

The actual expression used to calculate the aggregation should be an instance of `java.lang.Number`. The `SUM` statement always returns a `java.lang.Number` as the result and, depending on how big the value is and whether it has a decimal component or not, the returned type could be an instance of `java.lang.Integer`, `java.lang.Long`, `java.lang.Float` or `java.lang.Double`; you should take this into consideration when executing the query: if an overflow occurs while computing the `SUM` function because the value is higher than `Double.MAX_VALUE` ($(2 - 2^{-52}) * 2^{1023}$), the result will be incorrect.

The following are example `SUM` queries that return the summation of the entries ID.

```
SELECT SUM(ID)
FROM /exampleRegion
```

```
SELECT SUM(ID)
FROM /exampleRegion
WHERE ID > 0
```

```
SELECT SUM(ID)
FROM /exampleRegion
WHERE ID > 0 LIMIT 50
```

```
SELECT SUM(ID)
FROM /exampleRegion
WHERE ID > 0 AND status LIKE 'act%'
```

```
SELECT SUM(ID)
FROM /exampleRegion
WHERE ID IN SET(1,2,3,4,5)
```

The following `SUM` query returns the total summation of positive IDs grouped by status.

```
SELECT pf.status, SUM(pf.ID)
FROM /exampleRegion pf
WHERE pf.ID > 0
GROUP BY pf.status
```

The following `SUM` query uses the DISTINCT keyword and eliminates duplicates from the aggregation.

```
SELECT SUM(DISTINCT pf.ID)
FROM /exampleRegion pf, pf.positions.values pos
WHERE pf.ID > 0 OR pf.status = 'active' OR pos.secId = 'IBM'
```

The following `SUM` query returns the total aggregation of positive IDs grouped by status and sorted by the aggregation result in descending order.

```
SELECT pf.status, SUM(pf.ID) as sm
FROM /exampleRegion pf
WHERE pf.ID > 0
GROUP BY pf.status
ORDER BY sm DESC
```

AVG

The `AVG` keyword returns the arithmetic mean of the set formed by the selected expression. For partitioned regions, each node's buckets provide both a sum and the number of elements to the node executing the query (coordinator), such that a correct average may be computed.

The `AVG` keyword where the `DISTINCT` modifier is applied to the expression returns the arithmetic mean of the set of unique (distinct) values. For partitioned regions, the distinct values in a node's buckets are returned to the coordinator node, which can then calculate the average over the values that are unique across nodes, after eliminating duplicate values that come from separate nodes.

The actual expression used to calculate the aggregation should be an instance of `java.lang.Number`. The `AVG` statement always returns a `java.lang.Number` as the result and, depending on how big the value is and whether it has a decimal component or not, the returned type could be an instance of `java.lang.Integer`, `java.lang.Long`, `java.lang.Float` or `java.lang.Double`; you should take this into consideration when executing the query: if an overflow occurs while computing the `AVG` function because the value is higher than `Double.MAX_VALUE` ($(2 - 2^{-52}) * 2^{1023}$), or if an overflow occurs while computing the intermediate count because the amount of elements is higher than `Long.MAX_VALUE` ($2^{63} - 1$), the result will be incorrect.

The following are example `AVG` queries that calculate the average of the entries ID.

```
SELECT AVG(ID)
FROM /exampleRegion
```

```
SELECT AVG(ID)
FROM /exampleRegion
WHERE ID > 0
```

```
SELECT AVG(ID)
FROM /exampleRegion
WHERE ID > 0 LIMIT 50
```

```
SELECT AVG(ID)
FROM /exampleRegion
WHERE ID > 0 AND status LIKE 'act%'
```

```
SELECT AVG(ID)
FROM /exampleRegion
WHERE ID IN SET(1,2,3,4,5)
```

The following `AVG` query returns the average of positive IDs grouped by status.

```
SELECT pf.status, AVG(pf.ID)
FROM /exampleRegion pf
WHERE pf.ID > 0
GROUP BY pf.status
```

The following `AVG` query uses the `DISTINCT` keyword and eliminates duplicates from the aggregation.

```
SELECT AVG(DISTINCT pf.ID)
FROM /exampleRegion pf, pf.positions.values pos
WHERE pf.ID > 0 OR pf.status = 'active' OR pos.secId = 'IBM'
```

The following `AVG` query returns the average of positive IDs grouped by status and sorted by the calculation result in descending order.

```
SELECT pf.status, AVG(pf.ID) as sm
FROM /exampleRegion pf
WHERE pf.ID > 0
GROUP BY pf.status
ORDER BY sm DESC
```

OQL Syntax and Semantics

This section covers the following querying language features:

- **Supported Character Sets**
- **Supported Keywords**
- **Case Sensitivity**

- **Comments in Query Strings**
- **Query Language Grammar**
- **Operators**
- **Reserved Words**
- **Supported Literals**

Supported Character Sets

VMware GemFire query language supports the full ASCII and Unicode character sets.

Supported Keywords

Query Language Keyword	Description	Example
AND	Logical operator used to create complex expressions by combining two or more expressions to produce a Boolean result. When you combine two conditional expressions using the AND operator, both conditions must evaluate to true for the entire expression to be true.	See Operators
AS	Used to provide a label for a path expression so you can refer to the path by the label later.	See Aliases and Synonyms
AVG	Returns the average arithmetic mean of the set formed by the selected expression. The type of the expression must evaluate to a `java.lang.Number`.	See AVG
COUNT	Returns the quantity of values in the set formed by the selected expression.	See COUNT
DISTINCT	Restricts the select statement to unique results (eliminates duplicates).	See DISTINCT
ELEMENT	Query function. Extracts a single element from a collection or array. This function throws a `FunctionDomainException` if the argument is not a collection or array with exactly one element.	See Preset Query Functions
FROM	You can access any object or object attribute that is available in the current scope of the query.	See FROM Clause
GROUP BY	Allows you to summarize data, arranging or grouping results by identical field(s).	See GROUP BY
<HINT>	Keyword that instructs the query engine to prefer certain indexes.	See Using Query Index Hints
IMPORT	Used to establish the name for a class in a query.	See IMPORT Statement
IN	The IN expression is a Boolean indicating whether one expression is present inside a collection of expressions of a compatible type.	See IN and SET
IS_DEFINED	Query function. Returns TRUE if the expression does not evaluate to UNDEFINED. Inequality queries include undefined values in their query results. With the IS_DEFINED function, you can limit results to only those elements with defined values.	See Preset Query Functions

Query Language Keyword	Description	Example
IS_UNDEFINED	Query function. Returns TRUE if the expression evaluates to UNDEFINED. With the exception of inequality queries, most queries do not include undefined values in their query results. The IS_UNDEFINED function allows undefined values to be included, so you can identify elements with undefined values.	See Preset Query Functions
LIMIT	Limits the number of returned results. If you use the limit keyword, you cannot also run operations on the query result set that perform any kind of summary activities. For example trying to run add or addAll or a SelectResult from a query with a LIMIT clause throws an exception.	See LIMIT
LIKE	LIKE can be used to mean 'equals to', or if you terminate the string with a wildcard character (`%`), it behaves like 'starts with'. Note that the wildcard can only be used at the end of the comparison string. You can escape the wildcard character to represent the `%` character. You can also use the LIKE predicate if an index is present.	See LIKE
MIN	Returns the minimum or smallest value from the selected expression. The type of the expression must evaluate to a `java.lang.Comparable`.	See MIN
MAX	Returns the maximum or largest value from the selected expression. The type of the expression must evaluate to a `java.lang.Comparable`.	See MAX
NOT	The example returns the set of portfolios that have positions. Note that NOT cannot use an index.	See Operators
NVL	Returns expr2 if expr1 is null. The expressions can be query parameters (bind arguments), path expressions, or literals.	See Preset Query Functions
OR	If an expression uses both AND and OR operators, the AND expression has higher precedence than OR.	See Operators
ORDER BY	Allows you to order query results (either in ascending or descending order).	See ORDER BY
SELECT	Allows you to filter data from the collection of object(s) returned by a WHERE search operation.	See SELECT Statement
SET	Specifies a collection of values that can be compared to the returned values of query.	See IN and SET
SUM	Returns the summation over the set formed by the selected expression. The type of the expression must evaluate to a `java.lang.Number`.	See SUM
<TRACE>	Enables debugging on the following query string.	See Query Debugging
TO_DATE	Returns a Java Data class object. The arguments must be String S with date_str representing the date and format_str representing the format used by date_str. The format_str you provide is parsed using java.text.SimpleDateFormat.	See Preset Query Functions
TYPE	Specifying object type in the FROM clause helps the query engine to process the query at optimal speed.	See Object Typing
WHERE	Resolves to a collection of objects. The collection is then available for iteration in the query expressions that follow in the WHERE clause.	See WHERE Clause

Case Sensitivity

Query language keywords such as SELECT, NULL, DATE, and <TRACE> are case-insensitive. Identifiers such as attribute names, method names, and path expressions are case-sensitive.

In terms of query string and region entry matching, if you want to perform a case-insensitive search on a particular field, you can use the Java String class `toUpperCase` and `toLowerCase` methods in your query. For example:

```
SELECT entry.value FROM /exampleRegion.entries entry WHERE entry.value.toUpperCase LIK
E '%BAR%'
```

or

```
SELECT * FROM /exampleRegion WHERE foo.toLowerCase LIKE '%bar%'
```

Comments in Query Strings

Comment lines being with `--` (double dash). Comment blocks begin with `/*` and end with `*/`. For example:

```
SELECT * --my comment
FROM /exampleRegion /* here is
a comment */ WHERE status = ‘active’
```

Query Language Grammar

Language Grammar

Notation used in the grammar: n
A nonterminal symbol that has to appear at some place within the grammar on the left side of a rule. All nonterminal symbols have to be derived to be terminal symbols.

t
A terminal symbol (shown in italic bold).

x y
x followed by y

x | y
x or y

(x | y)
x or y

[x]
x or empty

{ x }
A possibly empty sequence of x.

comment
descriptive text

Grammar list:

```
symbol ::= expression
query_program ::= [ imports semicolon ] query [semicolon]
imports ::= import { semicolon import }
import ::= IMPORT qualifiedName [ AS identifier ]
query ::= selectExpr | expr
selectExpr ::= SELECT DISTINCT projectionAttributes fromClause [ whereClause ]
projectionAttributes ::= * | projectionList
projectionList ::= projection { comma projection }
projection ::= field | expr [ AS identifier ]
field ::= identifier colon expr
fromClause ::= FROM iteratorDef { comma iteratorDef }
iteratorDef ::= expr [ [ AS ] identifier ] [ TYPE identifier ] | identifier IN expr [
TYPE identifier ]
whereClause ::= WHERE expr
expr ::= castExpr
castExpr ::= orExpr | left_paren identifier right_paren castExpr
orExpr ::= andExpr { OR andExpr }
andExpr ::= equalityExpr { AND equalityExpr }
equalityExpr ::= relationalExpr { ( = | <> | != ) relationalExpr }
relationalExpr ::= additiveExpr { ( < | <= | > | >= ) additiveExpr }
additiveExpr ::= multiplicativeExpr { (+ | -) multiplicativeExpr }
multiplicativeExpr ::= inExpr { (MOD | % | / | *) inExpr}
inExpr ::= unaryExpr { IN unaryExpr }
unaryExpr ::= [ NOT ] unaryExpr
postfixExpr ::= primaryExpr { left_bracket expr right_bracket }
       | primaryExpr { dot identifier [ argList ] }
argList ::= left_paren [ valueList ] right_paren
qualifiedName ::= identifier { dot identifier }
primaryExpr ::= functionExpr
       | identifier [ argList ]
       | undefinedExpr
       | collectionConstruction
       | queryParam
       | literal
       | ( query )
       | region_path
functionExpr ::= ELEMENT left_paren query right_paren
       | NVL left_paren query comma query right_paren
       | TO_DATE left_paren query right_paren
undefinedExpr ::= IS_UNDEFINED left_paren query right_paren
       | IS_DEFINED left_paren query right_paren
collectionConstruction ::= SET left_paren [ valueList ] right_paren
valueList ::= expr { comma expr }
queryParam ::= $ integerLiteral
region_path ::= forward_slash region_name { forward_slash region_name }
region_name ::= name_character { name_character }
identifier ::= letter { name_character }
literal ::= booleanLiteral
       | integerLiteral
       | longLiteral
       | doubleLiteral
       | floatLiteral
       | charLiteral
       | stringLiteral
       | dateLiteral
       | timeLiteral
       | timestampLiteral
       | NULL
       | UNDEFINED
```

```
booleanLiteral ::= TRUE | FALSE
integerLiteral ::= [ dash ] digit { digit }
longLiteral ::= integerLiteral L
floatLiteral ::= [ dash ] digit { digit } dot digit { digit } [ ( E | e ) [ plus | das
h ] digit { digit } ] F
doubleLiteral ::= [ dash ] digit { digit } dot digit { digit } [ ( E | e ) [ plus | da
sh ] digit { digit } ] [ D ]
charLiteral ::= CHAR single_quote character single_quote
stringLiteral ::= single_quote { character } single_quote
dateLiteral ::= DATE single_quote integerLiteral dash integerLiteral dash integerLiter
al single_quote
timeLiteral ::= TIME single_quote integerLiteral colon
       integerLiteral colon integerLiteral single_quote
timestampLiteral ::= TIMESTAMP single_quote
       integerLiteral dash integerLiteral dash integerLiteral integerLiteral colon
       integerLiteral colon
       digit { digit } [ dot digit { digit } ] single_quote
letter ::= any unicode letter
character ::= any unicode character except 0xFFFF
name_character ::= letter | digit | underscore
digit ::= any unicode digit
```

The expressions in the following are all terminal characters:

```
dot ::= .
left_paren ::= (
right_paren ::= )
left_bracket ::= [
right_bracket ::= ]
single_quote ::= ’
underscore ::= _
forward_slash ::= /
comma ::= ,
semicolon ::= ;
colon ::= :
dash ::= -
plus ::= +
```

Language Notes

- Query language keywords such as SELECT, NULL, and DATE are case-insensitive. Identifiers such as attribute names, method names, and path expressions are case-sensitive.
- Comment lines begin with -- (double dash).
- Comment blocks begin with /* and end with */.
- String literals are delimited by single-quotes. Embedded single-quotes are doubled.

  Examples:

  ```
  'Hello' value = Hello
  'He said, ''Hello''' value = He said, 'Hello'
  ```

- Character literals begin with the CHAR keyword followed by the character in single quotation marks. The single-quotation mark character itself is represented as `CHAR ''''` (with four single quotation marks).

- In the TIMESTAMP literal, there is a maximum of nine digits after the decimal point.

Operators

VMware GemFire supports comparison, logical, unary, arithmetic, map, index, dot, and right arrow operators.

Comparison Operators

Comparison operators compare two values and return the results, either TRUE or FALSE.

The following are supported comparison operators:

Operator	Meaning
<	less than
<=	less than or equal to
>	greater than
>=	greater than or equal to
=	equal to
!=	not equal to
<>	not equal to

Regarding equality and inequality operators:

- The equality and inequality operators have lower precedence than the other comparison operators.
- The equality and inequality operators can be used with null.
- Inequality queries return results for which the search field is UNDEFINED.
- To perform equality or inequality comparisons with UNDEFINED, use the IS_DEFINED and IS_UNDEFINED preset query functions instead of these comparison operators.

Logical Operators

The logical operators AND and OR allow you to create more complex expressions by combining expressions to produce a boolean result. When you combine two conditional expressions using the AND operator, both conditions must evaluate to true for the entire expression to be true. When you combine two conditional expressions using the OR operator, the expression evaluates to true if either one or both of the conditions are true. You can create complex expressions by combining multiple simple conditional expressions with AND and OR operators. When expressions use AND and OR operators, AND has higher precedence than OR.

Unary Operators

Unary operators operate on a single value or expression, and have lower precedence than comparison operators in expressions. VMware GemFire supports the unary operator NOT. NOT is the negation operator, which changes the value of the operand to its opposite. For example, if an expression evaluates to TRUE, NOT changes it to FALSE. The operand must be a boolean.

Arithmetic Operators

Arithmetic operators operate on two values or expressions. Any of the expected arithmetic exceptions may result, such as overflow or a divide by zero. `QueryInvocationTargetException` will be thrown, and `getCause()` will state `ArithmeticException`.

The following are supported arithmetic operators:

Operator	Meaning
+	addition
-	subtraction
*	multiplication
/	division
%	modulus
MOD	modulus

Map and Index Operators

Map and index operators access elements in key/value collections (such as maps and regions) and ordered collections (such as arrays, lists, and `String`s). The operator is represented by a set of square brackets (`[ ]`) immediately following the name of the collection. The mapping or indexing specification is provided inside these brackets.

Array, list, and `String` elements are accessed using an index value. Indexing starts from zero for the first element, 1 for the second element, and so on. If `myList` is an array, list, or `String` and `index` is an expression that evaluates to a non-negative integer, then `myList[index]` represents the (`index + 1`)th element of `myList`. The elements of a `String` are the list of characters that make up the string.

Map and region values are accessed by key using the same syntax. The key can be any `Object`. For a `Region`, the map operator performs a non-distributed `get` in the local cache only - with no use of `netSearch`. So `myRegion[keyExpression]` is the equivalent of `myRegion.getEntry(keyExpression).getValue`.

Dot, Right Arrow, and Forward Slash Operators

The dot operator (`.`) separates attribute names in a path expression, and specifies the navigation through object attributes. An alternate equivalent to the dot is the right arrow, (`->`). The forward slash is used to separate region names when navigating into subregions.

Reserved Words

Reserved Words

These words are reserved for the query language and may not be used as identifiers. The words with asterisk (`*`) after them are not currently used by VMware GemFire, but are reserved for future implementation.

```
abs*
all
and
andthen*
any*
array
as
asc
avg*
bag*
boolean
by
byte
char
collection
count
date
declare*
define*
desc
```

```
dictionary
distinct
double
element
enum*
except*
exists*
false
first*
flatten*
float
for*
from
group*
having*
import
in
int
intersect*
interval*
```

```
is_defined
is_undefined
last*
like
limit
list*
listtoset*
long
map
max*
min*
mod
nil
not
null
nvl
octet
or
order
```

```
orelse*
query*
select
set
short
some*
string
struct*
sum*
time
timestamp
to_date
true
type
undefine*
undefined
union*
unique*
where
```

To access any method, attribute, or named object that has the same name as a query language reserved word, enclose the name within double quotation marks.

Examples:

```
SELECT DISTINCT "type" FROM /portfolios WHERE status = 'active'
```

```
SELECT DISTINCT * FROM /region1 WHERE emps."select"() < 100000
```

Supported Literals

VMware GemFire supports the following literal types:

boolean
: A \`boolean\` value, either TRUE or FALSE

int and **long**
: An integer literal is of type \`long\` if has a suffix of the ASCII letter L. Otherwise it is of type \`int\`.

floating point
: A floating-point literal is of type \`float\` if it has a suffix of an ASCII letter \`F\`. Otherwise its type is \`double\`. Optionally, it can have a suffix of an ASCII letter \`D\`. A double or floating point literal can optionally include an exponent suffix of \`E\` or \`e\`, followed by a signed or unsigned number.

string

String literals are delimited by single quotation marks. Embedded single-quotation marks are doubled. For example, the character string `'Hello'` evaluates to the value `Hello`, while the character string `'He said, ''Hello'''` evaluates to `He said, 'Hello'`. Embedded newlines are kept as part of the string literal.

char

A literal is of type char if it is a string literal prefixed by the keyword `CHAR`, otherwise it is of type `string`. The `CHAR` literal for the single-quotation mark character is `CHAR` `''''` (four single quotation marks).

date

A `java.sql.Date` object that uses the JDBC format prefixed with the DATE keyword: `DATE yyyy-mm-dd`. In the `Date`, `yyyy` represents the year, `mm` represents the month, and `dd` represents the day. The year must be represented by four digits; a two-digit shorthand for the year is not allowed.

time

A `java.sql.Time` object that uses the JDBC format (based on a 24-hour clock) prefixed with the TIME keyword: `TIME hh:mm:ss`. In the `Time`, `hh` represents the hours, `mm` represents the minutes, and `ss` represents the seconds.

timestamp

A `java.sql.Timestamp` object that uses the JDBC format with a TIMESTAMP prefix: `TIMESTAMP yyyy-mm-dd hh:mm:ss.fffffffff` In the `Timestamp`, `yyyy-mm-dd` represents the `date`, `hh:mm:ss` represents the `time`, and `fffffffff` represents the fractional seconds (up to nine digits).

NIL

Equivalent alternative of `NULL`.

NULL

The same as `null` in Java.

UNDEFINED

A special literal, valid value for any data type, indicating that no value (not even NULL) has been designated for a given data item.

The Difference Between NULL and UNDEFINED

In OQL, as in Java, NULL is an assignable entity (an object) indicating “no value”.

In OQL, UNDEFINED is a type. There is no Java equivalent. In OQL search results, an UNDEFINED value can be returned in two cases:

- As the result of a search for a key or value that does not exist
- As the result of accessing an attribute of a null-valued attribute.

Searches for inequality return UNDEFINED values in their results.

Note that if you access an attribute that has an explicit value of NULL, then it is not UNDEFINED.

For example, if a query accesses the attribute `address.city` and `address` is NULL, the result is UNDEFINED. If the query accesses `address`, then the result is not UNDEFINED, it is NULL.

Comparing Values With java.util.Date

You can compare temporal literal values `DATE`, `TIME`, and `TIMESTAMP` with `java.util.Date` values. There is no literal for `java.util.Date` in the query language.

Type Conversion

The VMware GemFire query processor performs implicit type conversions and promotions under certain cases in order to evaluate expressions that contain different types. The query processor performs binary numeric promotion, method invocation conversion, and temporal type conversion.

Binary Numeric Promotion

The query processor performs binary numeric promotion on the operands of the following operators:

- Comparison operators <, <=, >, and >=
- Equality operators = and <>
- Binary numeric promotion widens the operands in a numeric expression to the widest representation used by any of the operands. In each expression, the query processor applies the following rules in the prescribed order until a conversion is made:
    1. If either operand is of type double, the other is converted to double
    2. If either operand is of type float, the other is converted to float
    3. If either operand is of type long, the other is converted to long
    4. Both operands are converted to type int char

Method Invocation Conversion

Method invocation conversion in the query language follows the same rules as Java method invocation conversion, except that the query language uses runtime types instead of compile time types, and handles null arguments differently than in Java. One aspect of using runtime types is that an argument with a null value has no typing information, and so can be matched with any type parameter. When a null argument is used, if the query processor cannot determine the proper method to invoke based on the non-null arguments, it throws an AmbiguousNameException

Temporal Type Conversion

The temporal types that the query language supports include the Java types java.util.Date , java.sql.Date , java.sql.Time , and java.sql.Timestamp , which are all treated the same and can be compared and used in indexes. When compared with each other, these types are all treated as nanosecond quantities.

Enum Conversion

Enums are not automatically converted. To use Enum values in query, you must use the toString method of the enum object or use a query bind parameter. See Enum Objects for more information.

Query Evaluation of Float.NaN and Double.NaN

Float.NaN and Double.NaN are not evaluated as primitives; instead, they are compared in the same manner used as the JDK methods Float.compareTo and Double.compareTo. See Double.NaN and Float.NaN Comparisons for more information.

Query Language Restrictions and Unsupported Features

At a high level, VMware GemFire does not support the following querying features:

- Indexes targeted for joins across more than one region are not supported
- Static method invocations. For example, the following query is invalid:

  ```
  SELECT DISTINCT * FROM /QueryRegion0 WHERE aDay = Day.Wednesday
  ```

- You cannot create an index on fields using Set/List types (Collection types) that are not comparable. The OQL index implementation expects fields to be Comparable. To workaround this, you can create a custom Collection type that implements Comparable.
- ORDER BY is only supported with DISTINCT queries.

In addition, there are some specific limitations on partitioned region querying. See Partitioned Region Query Restrictions.

Advanced Querying

This section includes advanced querying topics such as using query indexes, using query bind parameters, querying partitioned regions and query debugging.

- **Performance Considerations**

  This topic covers considerations for improving query performance.

- **Monitoring Low Memory When Querying**

  The query monitoring feature prevents out-of-memory exceptions from occurring when you execute queries or create indexes.

- **Timeouts for Long-Running Queries**

  Configure a timeout value for long running queries, such that they do not complete, and VMware GemFire throws an exception when a query runs for longer than the configured value.

- **Using Query Bind Parameters**

  Using query bind parameters in VMware GemFire queries is similar to using prepared statements in SQL where parameters can be set during query execution. This allows you to build a query once and execute it multiple times by passing the query conditions during run time.

- **Querying Partitioned Regions**

  VMware GemFire allows you to manage and store large amounts of data across distributed nodes using partitioned regions. The basic unit of storage for a partitioned region is a bucket, which resides on a VMware GemFire node and contains all the entries that map to a single hashcode. In a typical partitioned region query, the system distributes the query to all buckets across all nodes, then merges the result sets and sends back the query results.

- **Query Debugging**

  You can debug a specific query at the query level by adding the `<trace>` keyword before the query string that you want to debug.

Performance Considerations

This topic covers considerations for improving query performance.

Some general performance tips:

- Improve query performance whenever possible by creating indexes. See Tips and Guidelines on Using Indexes for some scenarios for using indexes.
- Use bind parameters for frequently used queries. When you use a bind parameter, the query is compiled once. This improves the subsequent performance of the query when it is re-run. See Using Query Bind Parameters for more details.
- When querying partitioned regions, execute the query using the FunctionService. This function allows you to target a particular node, which will improve performance greatly by avoiding query distribution. See Querying a Partitioned Region on a Single Node for more information.
- Use key indexes when querying data that has been partitioned by a key or field value. See Optimizing Queries on Data Partitioned by a Key or Field Value.
- The size of a query result set depends on the restrictiveness of the query and the size of the total data set. A partitioned region can hold much more data than other types of regions, so there is more potential for larger result sets on partitioned region queries. This could cause the member receiving the results to run out of memory if the result set is very large.

Monitoring Low Memory When Querying

The query monitoring feature prevents out-of-memory exceptions from occurring when you execute queries or create indexes.

You enable this feature when you set a `critical-heap-percentage` attribute for the resource-manager element in the `cache.xml` file or by using the `cache.getResourceManager().setCriticalHeapPercentage(float heapPercentage)` API. When this feature is enabled and heap memory usage exceeds the threshold due to running a query or creating an index, the resource manager throws an exception and cancels the running query or index creation.

You can explicitly disable this feature by setting the system property `gemfire.cache.DISABLE_QUERY_MONITOR_FOR_LOW_MEMORY` to true.

When system memory is low, as determined by the critical heap percentage threshold defined in the `cache.xml` file or in the getResourceManager API, queries will throw a

`QueryExecutionLowMemoryException`. Any indexes that are in the process of being created will throw an `InvalidIndexException` with the message indicating the reason.

Partitioned Region Queries and Low Memory

Partitioned region queries are likely causes for out-of-memory exceptions. If query monitoring is enabled, partitioned region queries drop or ignore results that are being gathered by other servers if the executing server is low in memory.

Query-monitoring does not address a scenario in which a low-level collection is expanded while the partitioned region query is gathering results. For example, if a row is added and then causes a Java level collection or array to expand, it is possible to then encounter an out-of-memory exception. This scenario is rare and is only possible if the collection size itself expands before a low memory condition is met and then expands beyond the remaining available memory. As a workaround, in the event that you encounter this situation, you may be able to tune the system by additionally lowering the `critical-heap-percentage`.

Timeouts for Long-Running Queries

VMware GemFire can monitor and throw an exception when a query runs longer than a configured amount of time. This feature is enabled by setting the `critical-heap-percentage` attribute which detects that the JVM has too little heap memory.

The default query timeout is five hours. Set a different amount of time, in milliseconds, by specifying the system variable `gemfire.Cache.MAX_QUERY_EXECUTION_TIME`. A value of -1 explicitly disables the timeout.

When enabled, a query that runs longer than the configured timeout will be cancelled such that it does not finish, and VMware GemFire throws a `QueryExecutionTimeoutException`.

Using Query Bind Parameters

Using query bind parameters in VMware GemFire queries is similar to using prepared statements in SQL where parameters can be set during query execution. This allows user to build a query once and execute it multiple times by passing the query conditions during run time.

Query objects are thread-safe.

The use of query bind parameters is now supported in Client-to-Server queries.

The query parameters are identified by a dollar sign, $, followed by a digit that represents the parameter's position in the parameter array passed to the execute method. Counting begins at 1, so $1 references the first bound attribute, $2 the second attribute, and so on.

The Query interface provides an overloaded execute method that accepts parameters inside an Object array. See the Query.execute JavaDocs for more details.

The 0th element of the Object array is used for the first query parameter, and so on. If the parameter count or parameter types do not match the query specification, the execute method throws an exception. Specifically, if you pass in the wrong number of parameters, the method call throws a `QueryParameterCountInvalidException`. If a parameter object type is not compatible with what is expected, the method call throws a `TypeMismatchException`.

In the following example, the first parameter, the integer **2**, is bound to the first element in the object array. The second parameter, **active**, is bound to the second element.

Sample Code

```
// specify the query string
String queryString = "SELECT DISTINCT * FROM /exampleRegion p WHERE p.id = $1 and p.st
atus = $2";

QueryService queryService = cache.getQueryService();
Query query = queryService.newQuery(queryString);

// set query bind parameters
Object[] params = new Object[2];
params[0] = 2;
params[1] = "active";

// Execute the query locally. It returns the results set.
SelectResults results = (SelectResults) query.execute(params);

// use the results of the query; this example only looks at the size
 int size = results.size();
```

Using Query Bind Parameters in the Path Expression

Additionally the query engine supports the use of query bind parameter in place of a region path. When you specify a bind parameter in the query's FROM clause, the parameter's referenced value must be bound to a collection.

Examples:

The following query can be used on any collection by passing in the collection as a query parameter value. In this query you could pass in a Region object for $1, but not the String name of a region.

```
SELECT DISTINCT * FROM $1 p WHERE p.status = $2
```

Get all the keys from the region passed as a Region object for $1 for those entries whose name attribute is equal to the value passed for $2:

```
SELECT e.key FROM ($1).entrySet e WHERE e.value.name=$2"
```

Querying Partitioned Regions

VMware GemFire allows you to manage and store large amounts of data across distributed nodes using partitioned regions. The basic unit of storage for a partitioned region is a bucket, which resides on a VMware GemFire node and contains all the entries that map to a single hashcode. In a typical partitioned region query, the system distributes the query to all buckets across all nodes, then merges the result sets and sends back the query results.

The following list summarizes the querying functionality supported by VMware GemFire for partitioned regions:

- **Ability to target specific nodes in a query**. If you know that a specific bucket contains the data that you want to query, you can use a function to ensure that your query only runs the specific node that holds the data. This can greatly improve query efficiency. The ability to query data on a specific node is only available if you are using functions and if the function is executed on one single region. In order to do this, you need to use `Query.execute(RegionFunctionContext context)`. See the Java API and Querying a Partitioned Region on a Single Node for more details.
- **Ability to optimize partitioned region query performance using key indexes**. You can improve query performance on data that is partitioned by key or a field value by creating a key index and then executing the query using use `Query.execute(RegionFunctionContext context)` with the key or field value used as filter. See the Java API and Optimizing Queries on Data Partitioned by a Key or Field Value for more details.
- **Ability to perform equi-join queries between partitioned regions and between partitioned regions and replicated regions**. Join queries between partitioned region and between partitioned regions and replicated regions are supported through the function service. In order to perform equi-join operations on partitioned regions or partitioned regions and replicated regions, the partitioned regions must be colocated, and you need to use the need to use `Query.execute(RegionFunctionContext context)`. See the Java API and Performing an Equi-Join Query on Partitioned Regions for more details.
- **Using ORDER BY on Partitioned Regions**
- **Querying a Partitioned Region on a Single Node**
- **Optimizing Queries on Data Partitioned by a Key or Field Value**
- **Performing an Equi-Join Query on Partitioned Regions**
- **Partitioned Region Query Restrictions**

Using ORDER BY on Partitioned Regions

To execute a query with an ORDER BY clause on a partitioned region, the fields specified in the ORDER BY clause must be part of the projection list.

When an ORDER BY clause is used with a partition region query, the query is executed separately on each region host, the local query coordinator, and all remote members. The results are all gathered by the query coordinator. The cumulative result set is built by applying ORDER BY on the gathered results. If the LIMIT clause is also used in the query, ORDER BY and LIMIT are applied on each node before each node's results are returned to the coordinator. Then the clauses are applied to the cumulative result set to get the final result set, which is returned to the calling application.

Example:

```
// This query works because p.status is part of projection list
select distinct p.ID, p.status from /region p where p.ID > 5 order by p.status
// This query works providing status is part of the value indicated by *
select distinct * from /region where ID > 5 order by status
```

Querying a Partitioned Region on a Single Node

To direct a query to specific partitioned region node, you can execute the query within a function. Use the following steps:

1. Implement a function which executes a query using RegionFunctionContext.

```
/**
 * This function executes a query using its RegionFunctionContext
 * which provides a filter on data which should be queried.
 *
 */
public class MyFunction extends FunctionAdapter {

    private final String id;

    @Override
    public void execute(FunctionContext context) {

      Cache cache = CacheFactory.getAnyInstance();
      QueryService queryService = cache.getQueryService();

      String qstr = (String) context.getArguments();

      try {
        Query query = queryService.newQuery(qstr);

        //If function is executed on region, context is RegionFunctionContext
        RegionFunctionContext rContext = (RegionFunctionContext)context;

        SelectResults results = (SelectResults) query.execute(rContext)

        //Send the results to function caller node.
        context.getResultSender().sendResult((ArrayList) (results).asList());
        context.getResultSender().lastResult(null);

      } catch (Exception e) {
        throw new FunctionException(e);
      }
    }

    @Override
    public boolean hasResult() {
      return true;
    }

    @Override
    public boolean isHA() {
      return false;
    }


    public MyFunction(String id) {
      super();
      this.id = id;
    }

    @Override
    public String getId() {
      return this.id;
    }
  }
```

2. Decide on the data you want to query. Based on this decision, you can use `PartitionResolver` to configure the organization of buckets to be queried in the Partitioned Region.

   For example, let's say that you have defined the PortfolioKey class:

   ```
   public class PortfolioKey implements DataSerializable {
     private int id;
     private long startValidTime;
     private long endValidTime
     private long writtenTime

     public int getId() {
       return this.id;
     }
   ...
   }
   ```

   You could use the `MyPartitionResolver` to store all keys with the same ID in the same bucket. This `PartitionResolver` has to be configured at the time of Partition Region creation either declaratively using xml OR using APIs. See Configuring Partitioned Regions for more information.

   ```
   /** This resolver returns the value of the ID field in the key. With this resol
   ver,
    * all Portfolios using the same ID are colocated in the same bucket.
    */
   public class MyPartitionResolver implements PartitionResolver, Declarable {

      public Serializable getRoutingObject(EntryOperation operation) {
      return operation.getKey().getId();
   }
   ```

3. Execute the function on a client or any other node by setting the filter in the function call.

   ```
   /**
    * Execute MyFunction for query on specified keys.
    *
    */
   public class TestFunctionQuery {

     public static void main(String[] args) {

       ResultCollector rcollector = null;
       PortfolioKey portfolioKey1 = ...;

       //Filter data based on portfolioKey1 which is the key used in
       //region.put(portfolioKey1, portfolio1);
       Set filter = Collections.singleton(portfolioKey1);

       //Query to get all positions for portfolio ID = 1
       String qStr = "SELECT positions FROM /myPartitionRegion WHERE ID = 1";

       try {
   ```

```
      Function func = new MyFunction("testFunction");

      Region region = CacheFactory.getAnyInstance().getRegion("myPartitionRegio
n");

      //Function will be routed to one node containing the bucket
      //for ID=1 and query will execute on that bucket.
      rcollector = FunctionService
          .onRegion(region)
          .setArguments(qStr)
          .withFilter(filter)
          .execute(func);

      Object result = rcollector.getResult();

      //Results from one or multiple nodes.
      ArrayList resultList = (ArrayList)result;

      List queryResults = new ArrayList();

      if (resultList.size()!=0) {
        for (Object obj: resultList) {
          if (obj != null) {
            queryResults.addAll((ArrayList)obj);
          }
        }
      }
      printResults(queryResults);

    } catch (FunctionException ex) {
        getLogger().info(ex);
    }
  }
}
```

Optimizing Queries on Data Partitioned by a Key or Field Value

You can improve query performance on data that is partitioned by key or a field value by creating a key index and then executing the query using the `FunctionService` with the key or field value used as filter.

The following is an example how to optimize a query that will be run on data partitioned by region key value. In the following example, data is partitioned by the “orderId” field.

1. Create a key index on the orderId field. See Creating Key Indexes for more details.
2. Execute the query using the function service with orderId provided as the filter to the function context. For example:

```
/**
 * Execute MyFunction for query on data partitioned by orderId key
 *
 */
public class TestFunctionQuery {

  public static void main(String[] args) {
    Set filter =  new HashSet();
    ResultCollector rcollector = null;

    //Filter data based on orderId  = '12345'
    filter.add(12345);

    //Query to get all orders that match ID 12345 and amount > 1000
    String qStr = "SELECT * FROM /Orders WHERE orderId = '12345' AND amount > 1
000";

    try {
      Function func = new MyFunction("testFunction");

      Region region = CacheFactory.getAnyInstance().getRegion("myPartitionRegio
n");

      //Function will be routed to one node containing the bucket
      //for ID=1 and query will execute on that bucket.
      rcollector = FunctionService
          .onRegion(region)
          .setArguments(qStr)
          .withFilter(filter)
          .execute(func);

      Object result = rcollector.getResult();

      //Results from one or multiple nodes.
      ArrayList resultList = (ArrayList)result;

      List queryResults = new ArrayList();

      if (resultList.size()!=0) {
        for (Object obj: resultList) {
          if (obj != null) {
            queryResults.addAll((ArrayList)obj);
          }
        }
      }
      printResults(queryResults);

    } catch (FunctionException ex) {
        getLogger().info(ex);
    }
  }
}
```

Performing an Equi-Join Query on Partitioned Regions

In order to perform equi-join operations on partitioned regions or partitioned regions and replicated regions, you need to use the `query.execute` method and supply it with a function execution context. You need to use VMware GemFire's FunctionService executor because join operations are not yet directly supported for partitioned regions without providing a function execution context.

See Partitioned Region Query Restrictions for more information on partitioned region query limitations.

For example, let's say your equi-join query is the following:

```
SELECT DISTINCT * FROM /QueryRegion1 r1,
/QueryRegion2 r2 WHERE r1.ID = r2.ID
```

In this example QueryRegion2 is colocated with QueryRegion1, and both regions have same type of data objects.

On the server side:

```
 Function prQueryFunction1 = new QueryFunction();
 FunctionService.registerFunction(prQueryFunction1);

 public class QueryFunction extends FunctionAdapter {
    @Override
    public void execute(FunctionContext context) {
      Cache cache = CacheFactory.getAnyInstance();
      QueryService queryService = cache.getQueryService();
      ArrayList allQueryResults = new ArrayList();
      ArrayList arguments = (ArrayList)(context.getArguments());
      String qstr = (String)arguments.get(0);
      try {
           Query query = queryService.newQuery(qstr);
           SelectResults result = (SelectResults)query
             .execute((RegionFunctionContext)context);
           ArrayList arrayResult = (ArrayList)result.asList();
           context.getResultSender().sendResult((ArrayList)result.asList());
           context.getResultSender().lastResult(null);
              } catch (Exception e) {
               // handle exception
             }
       }
}
```

On the server side, `Query.execute()` operates on the local data of the partitioned region.

On the client side:

```
Function function = new QueryFunction();
String queryString = "SELECT DISTINCT * FROM /QueryRegion1 r1,
        /QueryRegion2 r2 WHERE r1.ID = r2.ID";
ArrayList argList = new ArrayList();
argList.add(queryString);
Object result = FunctionService.onRegion(CacheFactory.getAnyInstance()
     .getRegion("QueryRegion1" ))
     .setArguments(argList).execute(function).getResult();
ArrayList resultList = (ArrayList)result;
resultList.trimToSize();
List queryResults = null;
if (resultList.size() != 0) {
   queryResults = new ArrayList();
   for (Object obj : resultList) {
      if (obj != null ) {
      queryResults.addAll((ArrayList)obj);
         }
```

```
    }
}
```

On the client side, note that you can specify a bucket filter while invoking FunctionService.onRegion(). In this case, the query engine relies on FunctionService to direct the query to specific nodes.

Additional Notes on Using the Query.execute and RegionFunctionContext APIs

You can also pass multiple parameters (besides the query itself) to the query function by specifying the parameters in the client-side code (`FunctionService.onRegion(..).setArguments()`). Then you can handle the parameters inside the function on the server side using `context.getArguments`. Note that it does not matter which order you specify the parameters as long as you match the parameter handling order on the server with the order specified in the client.

Partitioned Region Query Restrictions

Query Restrictions in Partitioned Regions

Partitioned region queries function the same as non-partitioned region queries, except for the restrictions listed in this section. Partitioned region queries that do not follow these guidelines generate an `UnsupportedOperationException`.

- Join queries between partitioned region and between partitioned regions and replicated regions are supported through the function service only. Join queries partitioned regions are not supported through the client server API.
- You can run join queries on partitioned regions and on partitioned regions and replicated regions only if they are co-located. Equi-join queries are supported only on partitioned regions that are co-located and where the co-located columns are indicated in the WHERE clause of the query. In the case of multi-column partitioning, there should also be an AND clause in the WHERE specification. See Colocate Data from Different Partitioned Regions for more information on partitioned region co-location.
- Equi-join queries are allowed between partitioned regions and between partitioned regions and local replicated regions as long as the local replicated region also exists on all partitioned region nodes. To perform a join query on a partitioned region and another region (partitioned or not), you need to use the `query.execute` method and supply it with a function execution context. See Performing an Equi-Join Query on Partitioned Regions for an example.
- The query must be just a SELECT expression (as opposed to arbitrary OQL expressions), preceded by zero or more IMPORT statements. For example, this query is not allowed because it is not just a SELECT expression:

  ```
  // NOT VALID for partitioned regions
  (SELECT DISTINCT *FROM /prRgn WHERE attribute > 10).size
  ```

  This query is allowed:

  ```
  // VALID for partitioned regions
  SELECT DISTINCT *FROM /prRgn WHERE attribute > 10
  ```

- The SELECT expression itself can be arbitrarily complex, including nested SELECT expressions, as long as only one partitioned region is referenced.
- The partitioned region reference can only be in the first FROM clause iterator. Additional FROM clause iterators are allowed if they do not reference any regions (such as drilling down into the values in the partitioned region).
- The first FROM clause iterator must contain only one reference to the partitioned region (the reference can be a parameter, such as $1).
- The first FROM clause iterator cannot contain a subquery, but subqueries are allowed in additional FROM clause iterators.
- You can use ORDER BY on partitioned region queries, but the fields that are specified in the ORDER BY clause must be part of the projection list.
- If a partition region (or a bucket) being queried has been destroyed, the query is reattempted on the new primary for the destroyed bucket (if it exists). After certain number of attempts, a QueryException is thrown if all buckets (calculated at the startup of the query) cannot be queried.

Query Debugging

You can debug a specific query at the query level by adding the `<trace>` keyword before the query string that you want to debug.

Here is an example:

```
<trace> select * from /exampleRegion
```

You can also write:

```
<TRACE> select * from /exampleRegion
```

When the query is executed, VMware GemFire will log a message in `$GEMFIRE_DIR/system.log` with the following information:

```
[info 2011/08/29 11:24:35.472 PDT CqServer <main> tid=0x1] Query Executed in 9.619656
ms; rowCount = 99; indexesUsed(0) "select *  from /exampleRegion"
```

If you want to enable debugging for all queries, you can enable query execution logging by setting a System property on the command line during start-up:

```
gfsh>start server --name=server_name --J=-Dgemfire.Query.VERBOSE=true
```

Or you can set the property programmatically:

```
System.setProperty("gemfire.Query.VERBOSE","true");
```

As an example, let us say you have an EmployeeRegion that that contains Employee objects as values and the objects have public fields in them like ID and status.

```
Employee.java
Class Employee {
 public int ID;
 public String status;
 - - - - - -
 - - - - - -
}
```

In addition, you have created the following indexes for the region:

```
<index name="sampleIndex-1">
<functional from-clause="/test " expression="ID"/>
</index>
<index name="sampleIndex-2">
<functional from-clause="/test " expression="status"/>
</index>
```

After you have set `gemfire.Query.VERBOSE` to “true”, you could see the following debug messages in the logs after running queries on the EmployeeRegion or its indexes:

- If indexes are not used in the query execution, you would see a debug message like this:

  ```
  [info 2011/08/29 11:24:35.472 PDT CqServer <main> tid=0x1] Query Executed in 9.
  619656 ms; rowCount = 99; indexesUsed(0) "select * from /test k where ID > 0 an
  d status='active'"
  ```

- When single index is used in query execution, you might see a debug message like this:

  ```
  [info 2011/08/29 11:24:35.472 PDT CqServer <main> tid=0x1] Query Executed in 10
  1.43499 ms; rowCount = 199; indexesUsed(1):sampleIndex-1(Results: 199) "select
  count *   from /test k where ID > 0"
  ```

- When multiple indexes are used by a query, you might see a debug message like this:

  ```
  [info 2011/08/29 11:24:35.472 PDT CqServer <main> tid=0x1] Query Executed in 7
  9.43847 ms; rowCount = 199; indexesUsed(2):sampleIndex-2(Results: 100),sampleIn
  dex-1(Results: 199) "select * from /test k where ID > 0 OR status='active'"
  ```

In above log messages, the following information is provided:

- “rowCount” represents ResultSet size for the query.
- "indexesUsed(\n) " shows n indexes were used for finding the results of the query.
- Each index name and its corresponding results are reported respectively.
- The log can be identified with the original query string itself appended in the end.

Working with Indexes

The VMware GemFire query engine supports indexing. An index can provide significant performance gains for query execution.

A query run without the aid of an index iterates through every object in the collection. If an index is available that matches part or all of the query specification, the query iterates only over the indexed set, and query processing time can be reduced.

- **Tips and Guidelines on Using Indexes**

Optimizing your queries with indexes requires a cycle of careful planning, testing, and tuning. Poorly-defined indexes can degrade the performance of your queries instead of improving it. This section gives guidelines for index usage in the query service.

- **Creating, Listing and Removing Indexes**

  The VMware GemFire `QueryService` API provides methods to create, list and remove the index. You can also use `gfsh` command-line interface to create, list and remove indexes, and use cache.xml to create an index.

- **Creating Key Indexes**

  Creating a key index is a good way to improve query performance when data is partitioned using a key or a field value. You can create key indexes by using the `createKeyIndex` method of the QueryService or by defining the index in `cache.xml`. Creating a key index makes the query service aware of the relationship between the values in the region and the keys in the region.

- **Creating Hash Indexes**

  Hash indexes are deprecated. VMware GemFire supports the creation of hash indexes for the purposes of performing equality-based queries.

- **Creating Indexes on Map Fields ("Map Indexes")**

  To assist with the quick lookup of multiple values in a Map (or HashMap) type field, you can create an index (sometimes referred to as a "map index") on specific (or all) keys in that field.

- **Creating Multiple Indexes at Once**

  In order to speed and promote efficiency when creating indexes, you can define multiple indexes and then create them all at once.

- **Maintaining Indexes (Synchronously or Asynchronously) and Index Storage**

  Indexes are automatically kept current with the region data they reference. The region attribute `IndexMaintenanceSynchronous` specifies whether the region indexes are updated synchronously when a region is modified or asynchronously in a background thread.

- **Using Query Index Hints**

  You can use the hint keyword to allow VMware GemFire's query engine to prefer certain indexes.

- **Using Indexes on Single Region Queries**

  Queries with one comparison operation may be improved with either a key or range index, depending on whether the attribute being compared is also the primary key.

- **Using Indexes with Equi-Join Queries**

  Equi-join queries are queries in which two regions are joined through an equality condition in the WHERE clause.

- **Using Indexes with Overflow Regions**

  You can use indexes when querying on overflow regions; however, there are caveats.

- **Using Indexes on Equi-Join Queries using Multiple Regions**

To query across multiple regions, identify all equi-join conditions. Then, create as few indexes for the equi-join conditions as you can while still joining all regions.

- **Index Samples**

  This topic provides code samples for creating query indexes.

Tips and Guidelines on Using Indexes

Optimizing your queries with indexes requires a cycle of careful planning, testing, and tuning. Poorly-defined indexes can degrade the performance of your queries instead of improving it. This section gives guidelines for index usage in the query service.

When creating indexes, keep in mind the following:

- Indexes incur maintenance costs as they must be updated when the indexed data changes. An index that requires many updates and is not used very often may require more system resources than using no index at all.
- Indexes consume memory.
- Indexes have limited support on overflow regions. See Using Indexes with Overflow Regions for details.
- If you are creating multiple indexes on the same region, first define your indexes and then create the indexes all at once to avoid iterating over the region multiple times. See Creating Multiple Indexes at Once for details.

Tips for Writing Queries that Use Indexes

As with query processors that run against relational databases, the way a query is written can greatly affect execution performance. Among other things, whether indexes are used depends on how each query is stated. These are some of the things to consider when optimizing your VMware GemFire queries for performance:

- In general an index will improve query performance if the FROM clauses of the query and index match exactly.
- The query evaluation engine does not have a sophisticated cost-based optimizer. It has a simple optimizer which selects best index (one) or multiple indexes based on the index size and the operator that is being evaluated.
- For AND operators, you may get better results if the conditions that use indexes and conditions that are more selective come before other conditions in the query.
- Indexes are not used in expressions that contain NOT, so in a WHERE clause of a query, `qty >= 10` could have an index on `qty` applied for efficiency. However, `NOT(qty < 10)` could not have the same index applied.
- Whenever possible, provide a hint to allow the query engine to prefer a specific index. See Using Query Index Hints

Creating, Listing and Removing Indexes

The VMware GemFire `QueryService` API provides methods to create, list and remove the index. You can also use `gfsh` command-line interface to create, list and remove indexes, and use cache.xml to create an index.

Creating Indexes

Indexes can be created programmatically, by using the `gfsh` command line interface or by using cache.xml.

To create an index, use one of the following `QueryService` methods:

- `createIndex`. Creates the default type of index, a range index. Use this type of index if you will be writing queries that will be doing any kind of comparison operation besides an equality comparison.
- `createKeyIndex`. Creates a key index. See Creating Key Indexes for more information.
- **Deprecated.** `createHashIndex`. Creates a hash index. See Creating Hash Indexes for more information.
- `createDefinedIndexes`. Creates multiple indexes that were previously defined using `defineIndex`. See Creating Multiple Indexes at Once for more information.

The following sections provide examples of index creation:

Using gfsh:

```
gfsh> create index --name=myIndex --expression=status --region=/exampleRegion
gfsh> create index --name=myKeyIndex --type=key --expression=id --region=/exampleRegio
n
```

See Index Commands for more examples.

Using Java API:

```
QueryService qs = cache.getQueryService();
 qs.createIndex("myIndex", "status", "/exampleRegion");
 qs.createKeyIndex("myKeyIndex", "id", "/exampleRegion");
```

Using cache.xml:

```
<region name=exampleRegion>
 <region-attributes . . . >
 </region-attributes>
 <index name="myIndex" from-clause="/exampleRegion" expression="status"/>
 <index name="myKeyIndex" from-clause="/exampleRegion" expression="id" key-index="tru
e"/>
 ...
</region>
```

Note: If you do not specify the type of index in cache.xml, the type defaults to “range”.

Listing Indexes

To retrieve a list of indexes from the cache or region, use the `QueryService.getIndexes` method or the `gfsh` command line interface.

Using gfsh:

```
gfsh> list indexes
gfsh> list indexes --with-stats
```

Using Java API:

```
QueryService qs = cache.getQueryService();
 qs.getIndexes(); //returns a collection of all indexes in the cache
 qs.getIndexes(exampleRegion); //returns a collection of all indexes in exampleRegion
 qs.getIndexes(exampleRegion, myKeyIndex); //returns the index named myKeyIndex from t
he exampleRegion
```

Removing Indexes

To remove an index or all indexes from the cache or region, use the `QueryService.removeIndexes` method or the `gfsh` command line interface.

Using gfsh:

```
gfsh> destroy index
gfsh> destroy index --name=myIndex
gfsh> destroy index --region=/exampleRegion
```

Using Java API:

```
QueryService qs = cache.getQueryService();
 qs.removeIndexes(); //removes all indexes from the cache
 qs.removeIndexes(myKeyIndex); //removes the index named myKeyIndex
 qs.removeIndexes(exampleRegion); //removes all indexes from the exampleRegion
```

Creating Key Indexes

Creating a key index is a good way to improve query performance when data is partitioned using a key or a field value. You can create key indexes by using the `createKeyIndex` method of the QueryService or by defining the index in `cache.xml`. Creating a key index makes the query service aware of the relationship between the values in the region and the keys in the region.

The FROM clause for a primary key index must be just a region path. The indexed expression is an expression that, when applied to an entry value, produces the key. For example, if a region has Portfolios as the values and the keys are the id field of the Portfolios region, the indexed expression is id.

You can then use the FunctionService (using the partitioned key as a filter passed to the function and as part of the query equality condition) to execute the query against the indexed data. See Optimizing Queries on Data Partitioned by a Key or Field Value for more details.

There are two issues to note with key indexes:

- The key index is not sorted. Without sorting, you can only do equality tests. Other comparisons are not possible. To obtain a sorted index on your primary keys, create a functional index on the attribute used as the primary key.

- The query service is not automatically aware of the relationship between the region values and keys. For this, you must create the key index.

Note: Using a key-index with an explicit type=‘range’ in the cache.xml will lead to an exception. Key indexes will not be used in ‘range’ queries.

Examples of Creating a Key Index

Using Java API:

```
QueryService qs = cache.getQueryService();
 qs.createKeyIndex("myKeyIndex", "id", "/exampleRegion");
```

Using gfsh:

```
gfsh> create index --name=myKeyIndex --expression=id --region=/exampleRegion
```

Using cache.xml:

```
<region name=exampleRegion>
 <region-attributes . . . >
 </region-attributes>
 <index name="myKeyIndex" from-clause="/exampleRegion" expression="id" key-index="tru
e"/>
 ...
</region>
```

Note: If you do not specify the type of index when defining indexes using cache.xml, the type defaults to “range”.

Creating Hash Indexes

Hash indexes are deprecated. VMware GemFire supports the creation of hash indexes for the purpose of performing equality-based queries.

Hash Index Performance

The performance of put operations and recovery time when using a hash index will be worse than a range index. Queries are expected to be slower due to the implementation of the hash index and the cost of recalculating the key on request. A hash index can improve the memory usage of the index. So, the trade-off of the hash index space savings must be weighed against the performance penalty it imposes. If memory usage is not a concern, a range index is recommended.

Consider the memory usage when an index contains string fields. Copies of the strings are included in the index. With hash indexes, indexed expressions are canonicalized and stored in the index as pointers to the objects residing in the region, thereby using less memory. Tests achieved as high as a 30% reduction in memory footprint, but the savings depend on the keys and data being used.

Performance Considerations

Limitations

The following limitations must be considered when creating hash indexes:

- You can only use hash indexes with equals and not equals queries.
- Hash index maintenance will be slower than the other indexes due to synchronized add methods.
- Hash indexes cannot be maintained asynchronously. If you attempt to create a hash index on a region with asynchronous set as the maintenance mode, an exception will be thrown.
- You cannot use hash indexes for queries with multiple iterators or nested collections.
- Using a hash index will degrade put operation performance and recovery time substantially. If memory is not a concern, use a range index instead of a hash index.

Examples of Creating a Hash Index

Hash indexes are deprecated.

Using the Java API:

```
QueryService qs = cache.getQueryService();
 qs.createHashIndex("myHashIndex", "mktValue", "/exampleRegion");
```

Using gfsh:

```
gfsh> create index --name=myHashIndex --expression=mktValue --region=/exampleRegion
  --type=hash
```

Using cache.xml:

```
<region name=exampleRegion>
 <region-attributes . . . >
 </region-attributes>
 <index name="myHashIndex" from-clause="/exampleRegion p" expression="p.mktValue" type
="hash"/>
 ...
</region>
```

Creating Indexes on Map Fields ("Map Indexes")

To assist with the quick lookup of multiple values in a Map (or HashMap) type field, you can create an index (sometimes referred to as a “map index”) on specific (or all) keys in that field.

For example, you could create a map index to support the following query:

```
SELECT * FROM /users u WHERE u.name['first'] = 'John' OR u.name['last'] = 'Smith'
```

The map index extends regular range indexes created on single key by maintaining indexes for other specified keys, or for all keys if `*` is used. The underlying structure of the map index can be thought of as a wrapper around all these indexes.

The following Java code samples provide examples of how to create a map index:

```
QueryService qs = cache.getQueryService();

//This will create indexes for for keys 'PVTL' and 'VMW'
qs.createIndex("indexName", "p.positions['PVTL', 'VMW']", "/portfolio p");
```

```
QueryService qs = cache.getQueryService();

//This will create indexes for all keys
qs.createIndex("indexName", "p.positions[*]", "/portfolio p");
```

In gfsh, the equivalents are:

```
gfsh>create index --name="IndexName" --expression="p.positions['PVTL', 'VMW']" --regio
n="/portfolio p"

gfsh>create index --name="IndexName" --expression="p.positions[*]" --region="/portfoli
o p"
```

In order to create or query a map index, you must use the bracket notation to list the map field keys you wish to index or query. For example: `[*]`, `['keyX1','keyX2’]`. Note that using `p.pos.get('keyX1')` will not create or query the map index.

Note: You can still query against Map or HashMap fields without querying against a map index. For example, you can always create a regular range query on a single key in any Map or HashMap field. However, note that subsequent query lookups will be limited to a single key.

Creating Multiple Indexes at Once

In order to speed and promote efficiency when creating indexes, you can define multiple indexes and then create them all at once.

Defining multiple indexes before creating them speeds up the index creation process by iterating over region entries only once.

You can define multiple indexes of different types at once by specifying the `--type` parameter at definition time.

To define multiple indexes, you can use gfsh or the Java API:

gfsh example:

```
gfsh> define index --name=myIndex1 --expression=exp1 --region=/exampleRegion

gfsh> define index --name=myIndex2 --expression="c.exp2" --region="/exampleRegion e,
e.collection1 c"

gfsh> create defined indexes
```

If index creation fails, you may receive an error message in gfsh similar to the following:

```
gfsh>create defined indexes
Exception : org.apache.geode.cache.query.RegionNotFoundException ,
Message : Region ' /r3' not found: from  /r3Occurred on following members
1. india(s1:17866)<v1>:27809
```

Java API example:

```
 Cache cache = new CacheFactory().create();
    QueryService queryService = cache.getQueryService();
    queryService.defineIndex("name1", "indexExpr1", "regionPath1");
    queryService.defineIndex("name2", "indexExpr2", "regionPath2");
    queryService.defineKeyIndex("name4", "indexExpr4", "regionPath2");
    List<Index> indexes = queryService.createDefinedIndexes();
```

If one or more index population fails, VMware GemFire collect the Exceptions and continues to populate the rest of the indexes. The collected `Exceptions` are stored in a Map of index names and exceptions that can be accessed through `MultiIndexCreationException`.

Index definitions are stored locally on the `gfsh` client. If you want to create a new set of indexes or if one or more of the index creations fail, you might want to clear the definitions stored by using `clear defined indexes`command. The defined indexes can be cleared by using the Java API:

```
queryService.clearDefinedIndexes();
```

or gfsh:

```
gfsh> clear defined indexes
```

Maintaining Indexes (Synchronously or Asynchronously) and Index Storage

Indexes are automatically kept current with the region data they reference. The region attribute `IndexMaintenanceSynchronous` specifies whether the region indexes are updated synchronously when a region is modified or asynchronously in a background thread.

Index Maintenance Behavior

Asynchronous index maintenance batches up multiple updates to the same region key. The default mode is synchronous, since this provides the greatest consistency with region data.

See RegionFactory.setIndexMaintenanceSynchronous.

This declarative index creation sets the maintenance mode to asynchronous:

```
<region-attributes index-update-type="asynchronous">
</region-attributes>
```

Internal Index Structure and Storage

Indexes are stored either as compact or non-compact data structures based on the indexed expression (even if the index key type is the same.) For example, consider the following Passenger object:

```
Passenger {
   String name,
   Date travelDate,
   int age,
   Flight flt,
}
Flight {
   int flightId,
   String origin,
   String dest,
}
```

An index on the Passenger name field will have different memory space requirements in the cache than the Flight origin field even though they are both String field types. The internal data structure selected by VMware GemFire for index storage will depend on the field's level in the object. In this example, name is a top-level field and an index on name can be stored as a compact index. Since origin is a second-level field, any index that uses origin as the indexed expression will be stored as a non-compact index.

Compact Index

A compact index has simple data structures to minimize its footprint, at the expense of doing extra work at index maintenance. This index does not support the storage of projection attributes.

Compact indexes only support the creation of an index on a region path. In addition, the following conditions must be met:

- Index maintenance is synchronous.
- The indexed expression is a path expression.
- The FROM clause has only one iterator. This implies that there is only one value in the index for each region entry and it is directly on the region values (not supported with keys, entries).

Non-Compact Index

Used whenever a compact index cannot be used.

Using Query Index Hints

You can use the hint keyword to allow VMware GemFire's query engine to prefer certain indexes.

In cases where one index is hinted in a query, the query engine filters off the hinted index (if possible) and then iterates and filters from the resulting values.

Example:

```
<HINT 'IDIndex'> SELECT * FROM /Portfolios p WHERE p.ID > 10 AND p.owner = 'XYZ'
```

If multiple indexes are added as hints, then the query engine will try to use as many indexes as possible while giving a preference for the hinted indexes.

Example:

```
<HINT 'IDIndex', 'OwnerIndex'> SELECT * FROM /Portfolios p WHERE p.ID > 10 AND p.owner
```

```
= 'XYZ' AND p.value < 100
```

Using Indexes on Single Region Queries

Queries with one comparison operation may be improved with either a key or range index, depending on whether the attribute being compared is also the primary key.

If pkid is the key in the /exampleRegion region, creating a key index on pkid is the best choice as a key index does not have maintenance overhead. If pkid is not the key, a range index on pkid should improve performance.

```
SELECT DISTINCT * FROM /exampleRegion portfolio WHERE portfolio.pkid = '123'
```

With multiple comparison operations, you can create a range index on one or more of the attributes. Try the following:

1. Create a single index on the condition you expect to have the smallest result set size. Check performance with this index.
2. Keeping the first index, add an index on a second condition. Adding the second index may degrade performance. If it does, remove it and keep only the first index. The order of the two comparisons in the query can also impact performance. Generally speaking, in OQL queries, as in SQL queries, you should order your comparisons so the earlier ones give you the fewest results on which to run subsequent comparisons.

For this query, you would try a range index on name, age, or on both:

```
SELECT DISTINCT * FROM /exampleRegion portfolio WHERE portfolio.status = 'active' and
portfolio.ID > 45
```

For queries with nested levels, you may get better performance by drilling into the lower levels in the index as well as in the query.

This query drills down one level:

```
SELECT DISTINCT * FROM /exampleRegion portfolio, portfolio.positions.values positions
where positions.secId = 'AOL' and positions.MktValue > 1
```

Using Indexes with Equi-Join Queries

Equi-join queries are queries in which two regions are joined through an equality condition in the WHERE clause.

To use an index with an equi-join query:

1. Create an index for each side of the equi-join condition. The query engine can quickly evaluate the query's equi-join condition by iterating over the keys of the left-side and right-side indexes for an equality match.

   Note: Equi-join queries require regular indexes. Key indexes are not applied to equi-join queries.

   For this query:

```
SELECT DISTINCT inv.name, ord.orderID, ord.status
FROM /investors inv, /orders ord
WHERE inv.investorID = ord.investorID
```

Create two indexes:

FROM clause	Indexed expression
/investors inv	inv.investorID
/orders ord	ord.investorID

2. If there are additional, single-region queries in a query with an equi-join condition, create additional indexes for the single-region conditions only if you are able to create at least one such index for each region in the query. Any indexing on a subset of the regions in the query will degrade performance.

   For this example query:

```
SELECT DISTINCT *
FROM /investors inv, /securities sc, inv.heldSecurities inv_hs
    WHERE sc.status = "active"
    AND inv.name = "xyz"
    AND inv.age > 75
    AND inv_hs.secName = sc.secName
```

   Create the indexes for the equi-join condition:

FROM clause	Indexed expression
/investors inv, inv.heldSecurities inv_hs	inv_hs.secName
/securities sc	sc.secName

   Then, if you create any more indexes, create one on `sc.status` and one on `inv.age` or `inv.name` or both.

Using Indexes with Overflow Regions

You can use indexes when querying on overflow regions; however, there are caveats.

The following are caveats for querying overflow regions:

- You must use synchronous index maintenance for the region. This is the default maintenance setting.
- The index FROM clause must specify only one iterator, and it must refer to the keys or entry values. The index cannot refer to the region's entrySet.
- The index data itself is not stored on (overflowed to) disk .

Examples:

The following example index creation calls DO NOT work for overflow regions.

```
// This index will not work on an overflow region because there are two iterators in t
he FROM clause.
createIndex("secIdIndex", "b.secId","/portfolios pf, pf.positions.values b");
```

```
// This index will not work on an overflow region because the FROM clause specifies th
e entrySet
createIndex("indx1", "entries.value.getID", "/exampleRegion.entrySet() entries");
```

The following example indexes will work for overflow regions.

```
createIndex("pkidIndex", "p.pkid", "/Portfolios p");

createIndex("indx1", "ks.toString", "/portfolio.keySet() ks");
```

The same working examples in gfsh:

```
gfsh> create index -name="pkidIndex" --expression="p.pkid" --region="/Portfolios p"

gfsh> create index -name="indx1" --expression="ks.toString" --region="/portfolio.keySe
t() ks"
```

Using Indexes on Equi-Join Queries using Multiple Regions

To query across multiple regions, identify all equi-join conditions. Then, create as few indexes for the equi-join conditions as you can while still joining all regions.

If there are equi-join conditions that redundantly join two regions (in order to more finely filter the data, for example), then creating redundant indexes for these joins will negatively impact performance. Create indexes only on one equi-join condition for each region pair.

In this example query:

```
SELECT DISTINCT *
FROM /investors inv, /securities sc, /orders or,
inv.ordersPlaced inv_op, or.securities or_sec
    WHERE inv_op.orderID = or.orderID
    AND or_sec.secID = sc.secID
```

All conditions are required to join the regions, so you would create four indexes, two for each equi-join condition:

FROM clause	Indexed expression
/investors inv, inv.ordersPlaced inv_op	inv_op.orderID
/orders or, or.securities or_sec	or.orderID

FROM clause	Indexed expression
/orders or, or.securities or_sec	or_sec.secID
/securities sc	sc.secID

Adding another condition to the example:

```
SELECT DISTINCT *
FROM /investors inv, /securities sc, /orders or,
inv.ordersPlaced inv_op, or.securities or_sec, sc.investors sc_invs
    WHERE inv_op.orderID = or.orderID
    AND or_sec.secID = sc.secID
                AND inv.investorID = sc_invs.investorID
```

You would still only want to use four indexes in all, as that's all you need to join all of the regions. You would need to choose the most performant two of the following three index pairs:

FROM clause	Indexed expression
/investors inv, inv.ordersPlaced inv_op	inv_op.orderID
/orders or, or.securities or_sec	or.orderID

FROM clause	Indexed expression
/orders or, or.securities or_sec	or_sec.secID
/securities sc, sc.investors sc_invs	sc.secID

FROM clause	Indexed expression
/investors inv, inv.ordersPlaced inv_op	inv.investorID
/securities sc, sc.investors sc_invs	sc_invs.investorID

The most performant set is that which narrows the data to the smallest result set possible. Examine your data and experiment with the three index pairs to see which provides the best performance.

Index Samples

This topic provides code samples for creating query indexes.

```
 // Key index samples. The field doesn't have to be present.
createKeyIndex("pkidIndex","p.pkid1","/root/exampleRegion p");

createKeyIndex("Index4","ID","/portfolios");

// Simple index
createIndex("pkidIndex","p.pkid","/root/exampleRegion p");
createIndex("i", "p.status", "/exampleRegion p")
createIndex("i", "p.ID", "/exampleRegion p")
createIndex("i", "p.position1.secId", "/exampleRegion p"

// On Set type
 createIndex("setIndex","s","/root/exampleRegion p, p.sp s");

// Positions is a map
createIndex("secIdIndex","b.secId","/portfolios pf, pf.positions.values b");

//...
createIndex("i", "pf.collectionHolderMap[(pf.Id).toString()].arr[pf.ID]", "/exampleReg
ion pf")
createIndex("i", "pf.ID", "/exampleRegion pf", "pf.positions.values pos")
createIndex("i", "pos.secId", "/exampleRegion pf", "pf.positions.values pos")
createIndex("i", "e.value.getID()", "/exampleRegion.entrySet e")
createIndex("i", "e.value.ID", "/exampleRegion.entrySet e")

//...
createIndex("i", "entries.value.getID", "/exampleRegion.entrySet() entries")
createIndex("i", "ks.toString", "/exampleRegion.getKeys() ks")
createIndex("i", "key.status", "/exampleRegion.keys key")
createIndex("i", "secIds.length", "/exampleRegion p, p.secIds secIds")
createIndex("i", "secId", "/portfolios.asList[1].positions.values")
createIndex("i", "secId", "/portfolios['1'].positions.valules")

//Index on Map types
createIndex("i", "p.positions['key1']", "/exampleRegion p")
createIndex("i", "p.positions['key1','key2',key3',key7']", "/exampleRegion p")
createIndex("i", "p.positions[*]", "/exampleRegion p")
```

The following are some sample queries on indexes.

```
SELECT * FROM (SELECT * FROM /R2 m) r2, (SELECT * FROM  /exampleRegion e WHERE e.pkid
IN r2.sp) p

SELECT * FROM (SELECT * FROM /R2 m WHERE m.ID IN SET (1, 5, 10)) r2,
     (SELECT * FROM  /exampleRegion e WHERE e.pkid IN  r2.sp) p

//examples using position index in the collection
SELECT * FROM /exampleRegion p WHERE p.names[0] = 'aaa'

SELECT * FROM /exampleRegion p WHERE p.position3[1].portfolioId = 2

SELECT DISTINCT positions.values.toArray[0], positions.values.toArray[0], status
FROM /exampleRegion
```

Continuous Querying

Continuous querying continuously returns events that match the queries you set up.

- **How Continuous Querying Works**

  Clients subscribe to server-side events by using SQL-type query filtering. The server sends all events that modify the query results. CQ event delivery uses the client/server subscription framework.

- **Implementing Continuous Querying**

  Use continuous querying in your clients to receive continuous updates to queries run on the servers.

- **Managing Continuous Querying**

  This topic discusses CQ management options, CQ states, and retrieving initial result sets.

How Continuous Querying Works

Clients subscribe to server-side events by using SQL-type query filtering. The server sends all events that modify the query results. CQ event delivery uses the client/server subscription framework.

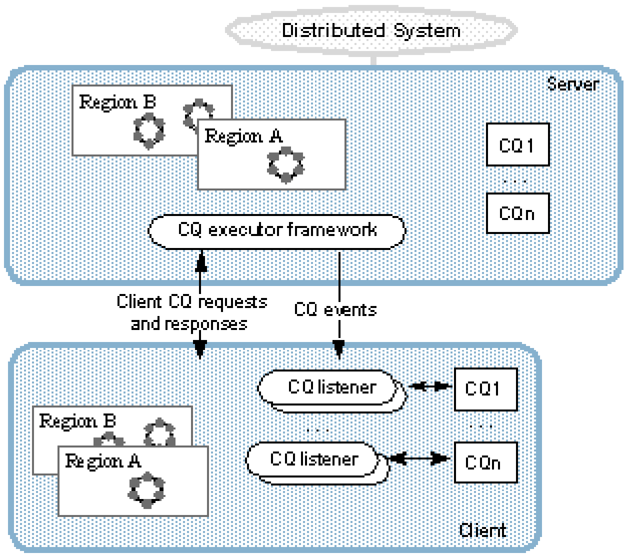
With CQ, the client sends a query to the server side for execution and receives the events that satisfy the criteria. For example, in a region storing stock market trade orders, you can retrieve all orders over a certain price by running a CQ with a query like this:

```
SELECT * FROM /tradeOrder t WHERE t.price > 100.00
```

When the CQ is running, the server sends the client all new events that affect the results of the query. On the client side, listeners programmed by you receive and process incoming events. For this example query on `/tradeOrder`, you might program a listener to push events to a GUI where higher-priced orders are displayed. CQ event delivery uses the client/server subscription framework.

Logical Architecture of Continuous Querying

Your clients can execute any number of CQs, with each CQ assigned any number of listeners.



Data Flow with CQs

CQs do not update the client region. This is in contrast to other server-to-client messaging like the updates sent to satisfy interest registration and responses to get requests from the client's `Pool`. CQs serve as notification tools for the CQ listeners, which can be programmed in any way your application requires.

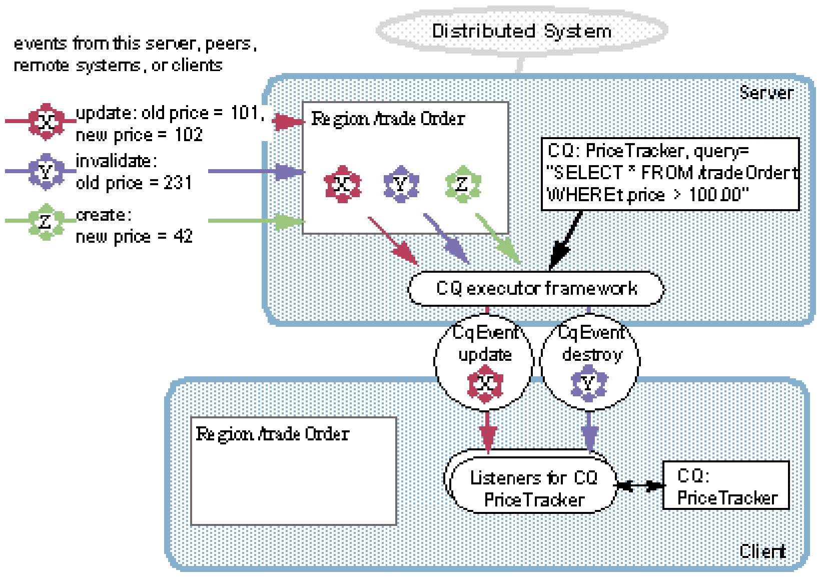
When a CQ is running against a server region, each entry event is evaluated against the CQ query by the thread that updates the server cache. If either the old or the new entry value satisfies the query, the thread puts a `CqEvent` in the client's queue. The `CqEvent` contains information from the original cache event plus information specific to the CQ's execution. Once received by the client, the `CqEvent` is passed to the `onEvent` method of all `CqListener`s defined for the CQ.

Here is the typical CQ data flow for entries updated in the server cache:

1. Entry events come to the server's cache from the server or its peers, distribution from remote sites, or updates from a client.
2. For each event, the server's CQ executor framework checks for a match with its running CQs.
3. If the old or new entry value satisfies a CQ query, a CQ event is sent to the CQ's listeners on the client side. Each listener for the CQ gets the event.

In the following figure:

- Both the new and old prices for entry X satisfy the CQ query, so that event is sent indicating an update to the query results.
- The old price for entry Y satisfied the query, so it was part of the query results. The invalidation of entry Y makes it not satisfy the query. Because of this, the event is sent indicating that it is destroyed in the query results.
- The price for the newly created entry Z does not satisfy the query, so no event is sent.


Distributed System
events from this server, peers, remote systems, or clients
update: old price = 101, new price = 102
invalidate: old price = 231
create: new price = 42
Server
Region /tradeOrder
CQ: PriceTracker, query= "SELECT * FROM /tradeOrder t WHERE t.price > 100.00"
CQ executor framework
CqEvent update
CqEvent destroy
Region /tradeOrder
Listeners for CQ PriceTracker
CQ: PriceTracker
Client

CQ Events

CQ events do not change your client cache. They are provided as an event service only. This allows you to have any collection of CQs without storing large amounts of data in your regions. If you need to persist information from CQ events, program your listener to store the information where it makes the most sense for your application.

The `CqEvent` object contains this information:

- Entry key and new value.
- Base operation that triggered the cache event in the server. This is the standard `Operation` class instance used for cache events in GemFire.
- `CqQuery` object associated with this CQ event.
- `Throwable` object, returned only if an error occurred when the `CqQuery` ran for the cache event. This is non-null only for `CqListener` onError calls.
- Query operation associated with this CQ event. This operation describes the change affected to the query results by the cache event. Possible values are:
    - `CREATE`, which corresponds to the standard database `INSERT` operation
    - `UPDATE`
    - `DESTROY`, which corresponds to the standard database DELETE operation

Region operations do not translate to specific query operations and query operations do not specifically describe region events. Instead, the query operation describes how the region event affects the query results.

Query operations based on old and new entry values	New value does not satisfy the query	New value satisfies the query
Old value does not satisfy the query	no event	`CREATE` query operation
Old value does satisfies the query	`DESTROY` query operation	`UPDATE` query operation

You can use the query operation to decide what to do with the `CqEvent` in your listeners. For example, a `CqListener` that displays query results on screen might stop displaying the entry, start displaying the entry, or update the entry display depending on the query operation.

Region Type Restrictions for CQs

You can only create CQs on replicated or partitioned regions. If you attempt to create a CQ on a non-replicated or non-partitioned region, you will receive the following error message:

```
The region <region name> specified in CQ creation is neither replicated nor partitione
d; only replicated or partitioned regions are allowed in CQ creation.
```

In addition, you cannot create a CQ on a replicated region with eviction setting of local-destroy since this eviction setting changes the region's data policy. If you attempt to create a CQ on this kind of region, you will receive the following error message:

```
CQ is not supported for replicated region: <region name> with eviction action: LOCAL_D
ESTROY
```

See also Configure Distributed, Replicated, and Preloaded Regions for potential issues with setting local-destroy eviction on replicated regions.

Implementing Continuous Querying

Use continuous querying in your clients to receive continuous updates to queries run on the servers.

Before you begin, you should be familiar with Querying and have your client/server system configured.

1. Configure the client pools you will use for CQs with `subscription-enabled` set to true.

   To have CQ and interest subscription events arrive as closely together as possible, use a single pool for everything. Different pools might use different servers, which can lead to greater differences in event delivery time.

2. Write your OQL query to retrieve the data you need from the server.

   The query must satisfy these CQ requirements in addition to the standard GemFire querying specifications: - The FROM clause must contain only a single region specification, with optional iterator variable. - The query must be a SELECT expression only, preceded by zero or more IMPORT statements. This means the query cannot be a statement such as `"/tradeOrder.name"` or `"(SELECT * from /tradeOrder).size".` - The CQ query cannot use: - Cross region joins - Drill-downs into nested collections - DISTINCT - Projections - Bind parameters - ORDER BY - GROUP BY - Aggregate Functions (MIN, MAX, AVG, SUM, COUNT) - The CQ query must be created on a partitioned or replicated region. See Region Type Restrictions for CQs.

The basic syntax for the CQ query is:

```
SELECT * FROM /fullRegionPath [iterator] [WHERE clause]
```

This example query could be used to get all trade orders where the price is over $100:

```
SELECT * FROM /tradeOrder t WHERE t.price > 100.00
```

3. Write your CQ listeners to handle CQ events from the server. Implement `org.apache.geode.cache.query.CqListener` in each event handler you need. In addition to your main CQ listeners, you might have listeners that you use for all CQs to track statistics or other general information.

   Note: Be especially careful if you choose to update your cache from your `CqListener`. If your listener updates the region that is queried in its own CQ and that region has a `Pool` named, the update will be forwarded to the server. If the update on the server satisfies the same CQ, it may be returned to the same listener that did the update, which could put your application into an infinite loop. This same scenario could be played out with multiple regions and multiple CQs, if the listeners are programmed to update each other's regions.

   This example outlines a `CqListener` that might be used to update a display screen with current data from the server. The listener gets the `queryOperation` and entry key and value from the `CqEvent` and then updates the screen according to the type of `queryOperation`.

```
// CqListener class
public class TradeEventListener implements CqListener
{
  public void onEvent(CqEvent cqEvent)
  {
    // org.apache.geode.cache Operation associated with the query op
    Operation queryOperation = cqEvent.getQueryOperation();
    // key and new value from the event
    Object key = cqEvent.getKey();
    TradeOrder tradeOrder = (TradeOrder)cqEvent.getNewValue();
    if (queryOperation.isUpdate())
    {
      // update data on the screen for the trade order . . .
    }
    else if (queryOperation.isCreate())
    {
      // add the trade order to the screen . . .
    }
    else if (queryOperation.isDestroy())
    {
      // remove the trade order from the screen . . .
    }
  }
  public void onError(CqEvent cqEvent)
  {
    // handle the error
  }
  // From CacheCallback public void close()
  {
    // close the output screen for the trades . . .
  }
}
```

When you install the listener and run the query, your listener will handle all of the CQ results.

4. If you need your CQs to detect whether they are connected to any of the servers that host its subscription queues, implement a `CqStatusListener` instead of a `CqListener`. `CqStatusListener` extends the current `CqListener`, allowing a client to detect when a CQ is connected and/or disconnected from the server(s). The `onCqConnected()` method will be invoked when the CQ is connected, and when the CQ has been reconnected after being disconnected. The `onCqDisconnected()` method will be invoked when the CQ is no longer connected to any servers.

   Taking the example from step 3, we can instead implement a `CqStatusListener`:

```
public class TradeEventListener implements CqStatusListener
{
  public void onEvent(CqEvent cqEvent)
  {
    // org.apache.geode.cache Operation associated with the query op
    Operation queryOperation = cqEvent.getQueryOperation();
    // key and new value from the event
    Object key = cqEvent.getKey();
    TradeOrder tradeOrder = (TradeOrder)cqEvent.getNewValue();
    if (queryOperation.isUpdate())
    {
      // update data on the screen for the trade order . . .
    }
    else if (queryOperation.isCreate())
    {
      // add the trade order to the screen . . .
    }
    else if (queryOperation.isDestroy())
    {
      // remove the trade order from the screen . . .
    }
  }
  public void onError(CqEvent cqEvent)
  {
    // handle the error
  }
  // From CacheCallback public void close()
  {
    // close the output screen for the trades . . .
  }

  public void onCqConnected() {
    //Display connected symbol
  }

  public void onCqDisconnected() {
    //Display disconnected symbol
  }
}
```

When you install the `CqStatusListener`, your listener will be able to detect its connection status to the servers that it is querying.

5. Program your client to run the CQ:
    1. Create a `CqAttributesFactory` and use it to set your `CqListener`s and `CqStatusListener`.
    2. Pass the attributes factory and the CQ query and its unique name to the `QueryService` to create a new `CqQuery`.
    3. Start the query running by calling one of the execute methods on the `CqQuery` object. You can execute with or without an initial result set.
    4. When you are done with the CQ, close it.

Continuous Query Implementation

```
// Get cache and queryService - refs to local cache and QueryService
// Create client /tradeOrder region configured to talk to the server

// Create CqAttribute using CqAttributeFactory
CqAttributesFactory cqf = new CqAttributesFactory();

// Create a listener and add it to the CQ attributes callback defined below
CqListener tradeEventListener = new TradeEventListener();
cqf.addCqListener(tradeEventListener);
CqAttributes cqa = cqf.create();
// Name of the CQ and its query
String cqName = "priceTracker";
String queryStr = "SELECT * FROM /tradeOrder t where t.price > 100.00";

// Create the CqQuery
CqQuery priceTracker = queryService.newCq(cqName, queryStr, cqa);

try
{  // Execute CQ, getting the optional initial result set
   // Without the initial result set, the call is priceTracker.execute();
  SelectResults sResults = priceTracker.executeWithInitialResults();
  for (Object o : sResults) {
       Struct s = (Struct) o;
       TradeOrder to = (TradeOrder) s.get("value");
       System.out.println("Intial result includes: " + to);
  }
}
  catch (Exception ex)
{
  ex.printStackTrace();
}
// Now the CQ is running on the server, sending CqEvents to the listener
. . .

// End of life for the CQ - clear up resources by closing
priceTracker.close();
```

With continuous queries, you can optionally implement:

- Highly available CQs by configuring your servers for high availability.

- Durable CQs by configuring your clients for durable messaging and indicating which CQs are durable at creation.

Managing Continuous Querying

This topic discusses CQ management options, CQ states, and retrieving initial result sets.

Using CQs from a RegionService Instance

If you are running durable client queues from the `RegionService` instance, stop and start the offline event storage for the client as a whole. The server manages one queue for the entire client process, so you need to request the stop and start of durable CQ event messaging for the cache as a whole, through the `ClientCache` instance. If you closed the `RegionService` instances, event processing would stop, but the server would continue to send events, and those events would be lost.

Stop with:

```
clientCache.close(true);
```

Start up again in this order:

1. Create `ClientCache` instance.
2. Create all `RegionService` instances. Initialize CQ listeners.
3. Call `ClientCache` instance `readyForEvents` method.

States of a CQ

A CQ has three possible states, which are maintained on the server. You can check them from the client through `CqQuery.getState`.

Query State	What does this mean?	When does the CQ reach this state?	Notes
STOPPED	The CQ is in place and ready to run, but is not running.	When CQ is first created and after being stopped from a running state.	A stopped CQ uses system resources. Stopping a CQ only stops the CQ event messaging from server to client. All server-side CQ processing continues, but new CQ events are not placed into the server's client queue. Stopping a CQ does not change anything on the client side (but, of course, the client stops receiving events for the CQ that is stopped).
RUNNING	The CQ is running against server region events and the client listeners are waiting for CQ events.	When CQ is executed from a stopped state.	This is the only state in which events are sent to the client.

Query State	What does this mean?	When does the CQ reach this state?	Notes
CLOSED	The CQ is not available for any further activities. You cannot rerun a closed CQ.	When CQ is closed by the client and when cache or connection conditions make it impossible to maintain or run.	The closed CQ does not use system resources.

CQ Management Options

You manage your CQs from the client side. All calls are executed only for the calling client's CQs.

Task	For a single CQ use …	For groups of CQs use …
Create a CQ	`QueryService.newCq`	N/A
Execute a CQ	`CqQuery.execute` and `CqQuery.executeWithInitialResults`	`QueryService.executeCqs`
Stop a CQ	`CqQuery.stop`	`QueryService.stopCqs`
Close a CQ	`CqQuery.close`	`QueryService.closeCqs`
Access a CQ	`CqEvent.getCq` and `QueryService.getCq`	`QueryService.getCq`
Modify CQ Listeners	`CqQuery.getCqAttributesMutator`	N/A
Access CQ Runtime Statistics	`CqQuery.getStatistics`	`QueryService.getCqStatistics`
Get all durable CQs registered on the server	N/A	`QueryService.getAllDurableCqsFromServer`

Managing CQs and Durable Clients Using gfsh

Using the `gfsh` command-line utility, you can perform the following actions:

- Close durable clients and durable client CQs. See close.
- List all durable CQs for a given durable client ID. See list.
- Show the subscription event queue size for a given durable client ID. See show subscription-queue-size.

Retrieving an Initial Result Set of a CQ

You can optionally retrieve an initial result set when you execute your CQ. To do this, execute the CQ with the `executeWithInitialResults` method. The initial `SelectResults` returned is the same that you would get if you ran the query ad hoc, by calling `QueryService.newQuery.execute` on the server cache, but with the key included. This example retrieves keys and values from an initial result set:

```
SelectResults cqResults = cq.executeWithInitialResults();
for (Object o : cqResults.asList()) {
  Struct s = (Struct) o; // Struct with Key, value pair
  Portfolio p = (Portfolio) s.get("value"); // get value from the Struct
  String id = (String) s.get("key"); // get key from the Struct
}
```

If you are managing a data set from the CQ results, you can initialize the set by iterating over the result set and then updating it from your listeners as events arrive. For example, you might populate a new screen with initial results and then update the screen from a CQ listener.

If a CQ is executed using the `ExecuteWithInitialResults` method, the returned result may already include the changes with respect to the event. This can arise when updates are happening on the region while CQ registration is in progress. The CQ does not block any region operation as it could affect the performance of the region operation. Design your application to synchronize between the region operation and CQ registration to avoid duplicate events from being delivered.

Transactions

This section describes VMware GemFire transactions. VMware GemFire offers an API for client applications that do transactional work. VMware GemFire implements optimistic transactions, with the familiar `begin`, `commit`, and `rollback` methods that implement the same operations as in relational database transactions methods.

- **Adherence to ACID Promises**

  This section explains the ways in which VMware GemFire's implementation of optimistic transactions provides ACID semantics.

- **Code Examples**

  An application-based transaction and a transaction embedded in a function provide examples to model.

- **Design Considerations**

  Designs that extend beyond the basics introduce other considerations. This section identifies and discusses how transactions interact with other aspects of the system.

Adherence to ACID Promises

This section introduces VMware GemFire transactions. VMware GemFire offers an API for client applications that do transactional work. VMware GemFire implements optimistic transactions, choosing the much higher transaction performance they offer over the slow, locking methods of a traditional relational database.

Optimistic transaction semantics are not identical to the Atomicity-Consistency-Isolation-Durability (ACID) semantics of a traditional relational database.

Atomicity

Atomicity is "all or nothing" behavior: a transaction completes successfully only when all of the operations it contains complete successfully. If problems occur during a transaction, perhaps due to other transactions with overlapping changes, the transaction cannot successfully complete until the problems are resolved.

Optimistic transactions provide atomicity and realize speed by using a reservation system, instead of using the traditional relational database technique of a two-phase locking of rows. The reservation prevents other, intersecting transactions from completing, allowing the commit to check for conflicts and to reserve resources in an all-or-nothing fashion prior to making changes to the data. After all changes have been made, locally and remotely, the reservation is released. With the reservation system, an intersecting transaction is simply discarded. The serialization of obtaining locks is avoided.

Consistency

Consistency requires that data written within a transaction must observe the key and value constraints established for the affected region. Note that validity of the transaction is the responsibility of the application.

Isolation

Isolation is the level at which transactional state is visible to system components. VMware GemFire transactions have repeatable read isolation. Once the committed value is read for a given key, it always returns that same value. If a write within a transaction deletes a value for a key that has already been read, subsequent reads return the transactional reference.

The default configuration isolates transactions at the process thread level. While a transaction is in progress, its changes are visible only inside the thread that is running the transaction. Other threads within that same process and threads in other processes cannot see changes until after the commit operation begins. After beginning the commit, the changes are visible in the cache, but other threads that access the changing data might see partial results of the transaction, leading to a dirty read. See Changing the Handling of Dirty Reads for how to change the default behavior.

Durability

Relational databases provide durability by using disk storage for recovery and transaction logging. VMware GemFire is optimized for performance and does not support on-disk durability for transactions.

See Allowing Transactions to Work on Persistent Regions for how to allow a transaction that operates on a persistent region in a non-durable way.

Code Examples

An application can run a transaction directly or invoke a function which contains a transaction. This section illustrates these two use cases with code fragments that demonstrate the proper way to program a transaction.

An expected use case operates on two regions within a transaction. For performance purposes the VMware GemFire transaction implementation requires that region entries of partitioned regions be colocated. See Custom-Partitioning and Colocating Data for details on how to colocate region entries.

Transaction within an Application

An application/client uses the `CacheTransactionManager` API. This most basic code fragment shows the structure of a transaction, with its `begin` to start the transaction, `commit` to end the transaction, and handling of exceptions that these methods may throw.

```
CacheTransactionManager txManager =
          cache.getCacheTransactionManager();

try {
    txManager.begin();
    // ... do transactional, region operations
    txManager.commit();
} catch (CommitConflictException conflict) {
    // ... do necessary work for a transaction that failed on commit
} finally {
    // All other exceptions will be handled by the caller.
    // Examples of some exceptions: the data is not colocated, a rebalance
    // interfered with the transaction, or the server is gone.
    // Any exception thrown by a method other than commit() needs
    // to do a rollback to avoid leaking the transaction state.
    if(txManager.exists()) {
        txManager.rollback();
    }
}
```

More details of a transaction appear in this next application/client code fragment example. In this typical transaction, the put operations must be atomic and two regions are involved.

In this transaction, a customer's purchase is recorded. The `cash` region contains each customer's cash balance available for making trades. The `trades` region records each customer's balance spent on trades.

If there is a conflict upon commit of the transaction, an exception is thrown, and this example tries again.

```
// inputs needed for this transaction; shown as variables for simplicity
final String customer = "Customer1";
final Integer purchase = 1000;

// region set up shown to promote understanding
Cache cache = new CacheFactory().create();
Pool pool = PoolManager.createFactory()
           .addLocator("localhost", LOCATOR_PORT)
           .create("pool-name");
Region<String, Integer> cash =
           cache.createClientRegionFactory(ClientRegionShortcut.PROXY)
           .setPoolName(pool.getName())
           .create("cash");
Region<String, Integer> trades =
           cache.createClientRegionFactory(ClientRegionShortcut.PROXY)
           .setPoolName(pool.getName())
           .create("trades");

// transaction code
CacheTransactionManager txManager = cache.getCacheTransactionManager();
boolean retryTransaction = false;
do {
  try {
    txManager.begin();

    // Subtract out the cost of the trade for this customer's balance
    Integer cashBalance = cash.get(customer);
    Integer newBalance = (cashBalance != null ? cashBalance : 0) - purchase;
    cash.put(customer, newBalance);

    // Add in the cost of the trade for this customer
    Integer tradeBalance = trades.get(customer);
    newBalance = (tradeBalance != null ? tradeBalance : 0) + purchase;
    trades.put(customer, newBalance);

    txManager.commit();
    retryTransaction = false;
  }
  catch (CommitConflictException conflict) {
    // entry value changed causing a conflict for this customer, so try again
    retryTransaction = true;
  } finally {
    // All other exceptions will be handled by the caller.
    // Any exception thrown by a method other than commit() needs
    // to do a rollback to avoid leaking the transaction state.
    if(txManager.exists()) {
      txManager.rollback();
    }
  }

} while (retryTransaction);
```

Design transactions such that any get operations are within the transaction. This causes those entries to be part of the transactional state, which is desired such that intersecting transactions can be detected and signal commit conficts.

Transaction within a Function

A transaction may be embedded in a function. The application invokes the function, and the function contains the transaction that does the `begin`, the region operations, and the `commit` or `rollback`.

This use of a function can have performance benefits. The performance benefit results from both the function and the region data residing on servers. As the function invokes region operations, those operations on region entries stay on the server, so there is no network round trip time to do get or put operations on region data.

This function example accomplishes atomic updates on a single region representing the quantity of products available in inventory. Doing this in a transaction prevents double allocating inventory for two orders placed simultaneously.

```
/**
 * Atomically reduce inventory quantity
 */
public class TransactionalFunction extends Function {

  /**
   * Returns true if the function had the requested quantity of
   * inventory and successfully completed the transaction to
   * record the reduced inventory that fulfills the order.
   */
  @Override
  public void execute(FunctionContext context) {
    RegionFunctionContext rfc = (RegionFunctionContext) context;
    Region<ProductId, Integer> inventoryRegion = rfc.getDataSet();

    CacheTransactionManager
        txManager = context.getCache().getCacheTransactionManager();

    // single argument will be a ProductId and a quantity
    ProductRequest request = (ProductRequest) rfc.getArguments();
    ProductId productRequested = request.getProductId();
    Integer qtyRequested = request.getQuantity();

    boolean success = false;

    do {
      boolean commitConflict = false;
      try {
        txManager.begin();

        Integer qtyAvailable = inventoryRegion.get(productRequested);
        if (qtyAvailable >= qtyRequested) {
          // enough inventory is available, so process request
          Integer remaining = qtyAvailable - qtyRequested;
          inventoryRegion.put(productRequested, remaining);
          txManager.commit();
          success = true;
        }

      } catch (CommitConflictException conflict) {
        // retry transaction, as another request on this same key succeeded,
        // so this transaction attempt failed
        commitConflict = true;
      } finally {
        // All other exceptions will be handled by the caller; however,
        // any exception thrown by a method other than commit() needs
        // to do a rollback to avoid leaking the transaction state.
        if(txManager.exists()) {
          txManager.rollback();
        }
      }

    } while (commitConflict);

    context.getResultSender().lastResult(success);
  }

  @Override
  public String getId() {
    return "TxFunction";
  }

  /**
   * Returning true causes this function to execute on the server
   * that holds the primary bucket for the given key. It can save a
   * network hop from the secondary to the primary.
   */
  @Override
  public boolean optimizeForWrite() {
    return true;
  }
}
```

The application-side details on function implementation are not covered in this example. The application sets up the function context and the argument. See the section on Function Execution for details on functions.

The function implementation needs to catch the commit conflict exception such that it can retry the entire transaction. The exception only occurs if another request for the same product intersected with this one, and that other request's transaction committed first.

The `optimizeForWrite` method is defined to cause the system to execute the function on the server that holds the primary bucket for the given key. It can save a network hop from the secondary to the primary.

Note that the variable `qtyAvailable` is a reference, because the `Region.get` operation returns a reference within this server-side code. Read Region Operations Return References for details and how to work around the implications of a reference as a return value when working with server code.

Design Considerations

Designs that incorporate more complex features introduce further considerations. This section discusses how transactions interact with other VMware GemFire features.

- **Colocate Partitioned Regions**
- **Region Operations Return References**
- **First Operation with Mixed Region Types**
- **Allowing Transactions to Work on Persistent Regions**
- **Mixing Transactions with Queries and Indexes**
- **Mixing Transactions with Eviction**
- **Mixing Transactions with Expiration**
- **Mixing Transactions with Non-transactional Operations**
- **Changing the Handling of Dirty Reads**

Colocate Partitioned Regions

For performance, transactions that operate on more than one partitioned region require that those partitioned regions colocate their entries. Colocate Data from Different Partitioned Regions describes how to colocate entries.

Region Operations Return References

For performance, server-invoked region operations return references to region entries. Any assignment to that reference changes the entry within the region. This subverts the system's ability to maintain consistency and the callback chain for handlers such as cache writers and cache loaders.

Changing an entry using a reference from within a transaction executing on a server has the same consistency issues, but is even worse, as the change will not be seen as part of the transactional state.

There are two ways to work with a reference: make a copy, or configure the system to return copies instead of references. There is a performance penalty to having the system return copies. Both ways are detailed in Copy on Read Behavior.

First Operation with Mixed Region Types

When more than one region participates in a transaction, and there is at least one partitioned and at least one replicated region, the code must do its first operation on the partitioned region to avoid a `TransactionDataNotColocatedException`. Write the transaction to do its first operation on a partitioned region, even if the operation will be spurious.

Allowing Transactions to Work on Persistent Regions

VMware GemFire's implementation of atomic transactions prohibits regions with persistence from participating in transactions. The invocation of a persistent region operation within a transaction throws an `UnsupportedOperationException` with an associated message of

```
Operations on persist-backup regions are not allowed because this thread
has an active transaction
```

An application that wishes to allow operations on a persistent region during a transaction can set this system property:

`-Dgemfire.ALLOW_PERSISTENT_TRANSACTIONS=true`

Setting this system property eliminates the exception. It does not change the fact that atomicity is not enforced for disk writes that occur with the commit of a transaction. A server crash during the commit may succeed in some, but not all of the disk writes.

Mixing Transactions with Queries and Indexes

Queries and query results reflect region state, and not any state or changes that occur within a transaction. Likewise, the contents and updates to an index do not intersect with any changes made within a transaction. Therefore, do not mix transactions with queries or indexed regions.

Mixing Transactions with Eviction

LRU eviction and transactions work well together. Any eviction operation on a region entry that is operated on from within a transaction is deferred until the transaction is committed. Further, because any entry touched by the transaction has had its LRU clock reset, eviction is not likely to choose those entries as victims immediately after the commit.

Mixing Transactions with Expiration

A transaction disables expiration on any region entries affected by the transaction.

Mixing Transactions with Non-transactional Operations

For best performance, non-transactional operations do not acquire the exclusive locks used to check for conflicts in a transaction. A transaction operating on the same data as a non-transactional actor is unable to detect the conflict caused by a non-transactional operation.

If using transactions, an application should adopt the policy of designating certain regions or sets of entries exclusively for transactional puts, updates, and deletions, so transactional entries will not be modified by non-transactional operations.

If other, non-transactional sources update the keys the transaction is modifying, the changes may intermingle with the transaction's changes. The other sources can include distributions from remote members, loading activities, and other direct cache modification calls from the same member. When this happens, after your commit finishes, the cache state may not be what you expected.

Changing the Handling of Dirty Reads

An application requiring a strict, but slower isolation model, such that dirty reads of transitional states are not allowed, should set a property and encapsulate read operations within the transaction. Configure this strict isolation model with the property:

```
-Dgemfire.detectReadConflicts=true
```

This property causes read operations to succeed only when they read a consistent pre- or post-transactional state. If not consistent, VMware GemFire throws a `CommitConflictException`.

Function Execution

A function is a body of code that resides on a server and that an application can invoke from a client or from another server without the need to send the function code itself. The caller can direct a data-dependent function to operate on a particular dataset, or can direct a data-independent function to operate on a particular server, member, or member group.

The function execution service provides solutions for a variety of use cases, including:

- An application needs to perform an operation on the data associated with a key. A registered server-side function can retrieve the data, operate on it, and put it back, with all processing performed locally to the server.
- An application needs to initialize some of its components once on each server, which might be used later by executed functions.
- A third-party service, such as a messaging service, requires initialization and startup.
- Any arbitrary aggregation operation requires iteration over local data sets that can be done more efficiently through a single call to the cache server.
- An external resource needs provisioning that can be done by executing a function on a server.

- How Function Execution Works
- Executing a Function in VMware GemFire

How Function Execution Works

Where Functions Are Executed

You can execute data-independent functions or data-dependent functions in VMware GemFire in the following places:

For Data-independent Functions

- **On a specific member or members:** Execute the function within a peer-to-peer cluster, specifying the member or members where you want to run the function by using `FunctionService` methods `onMember()` and `onMembers()`.
- **On a specific server or set of servers:** If you are connected to a cluster as a client, you can execute the function on a server or servers configured for a specific connection pool, or on a server or servers connected to a given cache using the default connection pool. For data-independent functions on client/server architectures, a client invokes `FunctionService` methods `onServer()` or `onServers()`. (See How Client/Server Connections Work for details regarding pool connections.)
- **On member groups or on a single member within each member group:** You can organize members into logical member groups. (See Configuring and Running a Cluster for more information about using member groups.) You can invoke a data independent function on all members in a specified member group or member groups, or execute the function on only one member of each specified member group.

For Data-dependent Functions

- **On a region:** If you are executing a data-dependent function, specify a region and, optionally, a set of keys on which to run the function. The method `FunctionService.onRegion()` directs a data-dependent function to execute on a specific region.

See the `org.apache.geode.cache.execute.FunctionService` Java API documentation for more details.

How Functions Are Executed

The following things occur when executing a function:

1. For security-enabled clusters, prior to executing the function, a check is made to see that that caller is authorized to execute the function. The required permissions for authorization are provided by the function's `Function.getRequiredPermissions()` method. See Authorization of Function Execution for a discussion of this method.
2. Given successful authorization, VMware GemFire invokes the function on all members where it needs to run. The locations are determined by the `FunctionService on*` method calls, region configuration, and any filters.

3. If the function has results, they are returned to the `addResult` method call in a `ResultCollector` object.
4. The originating member collects results using `ResultCollector.getResult`.

Highly Available Functions

Generally, function execution errors are returned to the calling application. You can code for high availability for `onRegion` functions that return a result, so VMware GemFire automatically retries a function if it does not execute successfully. You must code and configure the function to be highly available, and the calling application must invoke the function using the results collector `getResult` method.

When a failure (such as an execution error or member crash while executing) occurs, the system responds by:

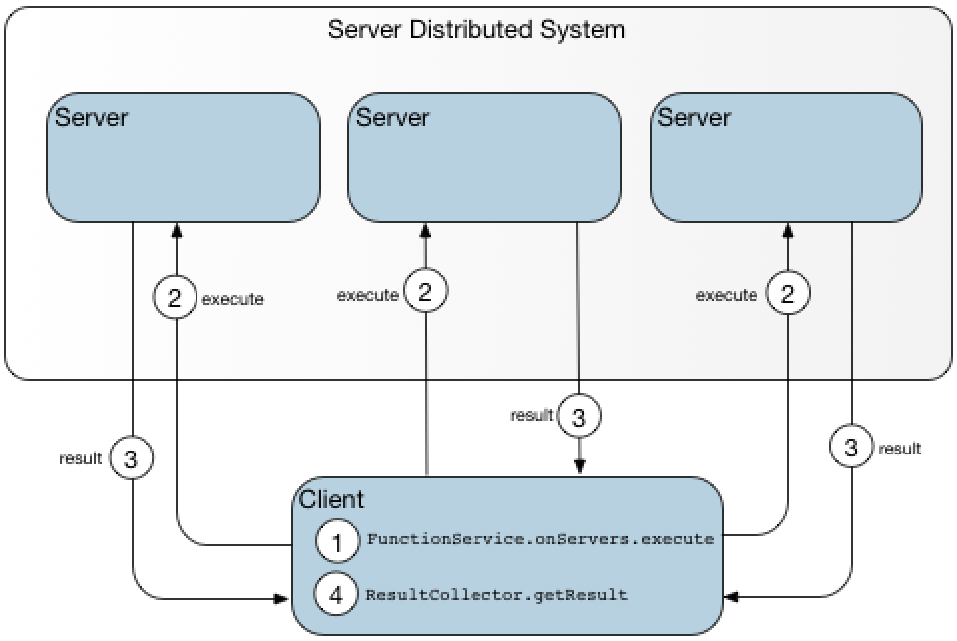
1. Waiting for all calls to return
2. Setting a boolean indicating a re-execution
3. Calling the result collector's `clearResults` method
4. Executing the function

For client regions, the system retries the execution according to `org.apache.geode.cache.client.Pool` `retryAttempts`. If the function fails to run every time, the final exception is returned to the `getResult` method.

The default number of retries is the total number of servers present. For member calls, if a function fails then the system will retry on each server only once or until no data remains in the system for the function to operate on. If the function fails on every server, then an exception will be returned to the user.

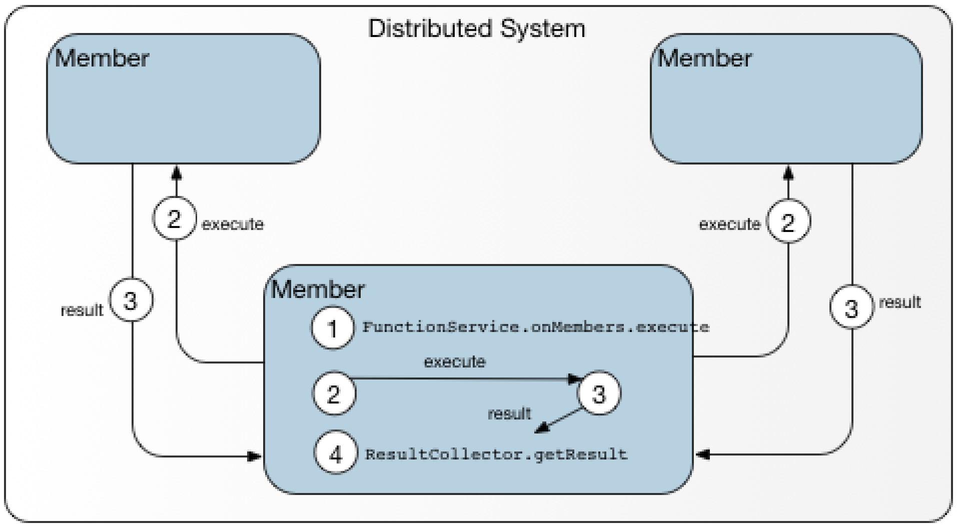
Function Execution Scenarios

This figure shows the sequence of events for a data-independent function invoked from a client on all available servers.


Server Distributed System
Server
Server
Server
2 execute
execute 2
execute 2
result 3
result 3
3 result
Client
1 FunctionService.onServers.execute
4 ResultCollector.getResult

The client contacts a locator to obtain host and port identifiers for each server in the cluster and issues calls to each server. As the instigator of the calls, the client also receives the call results.

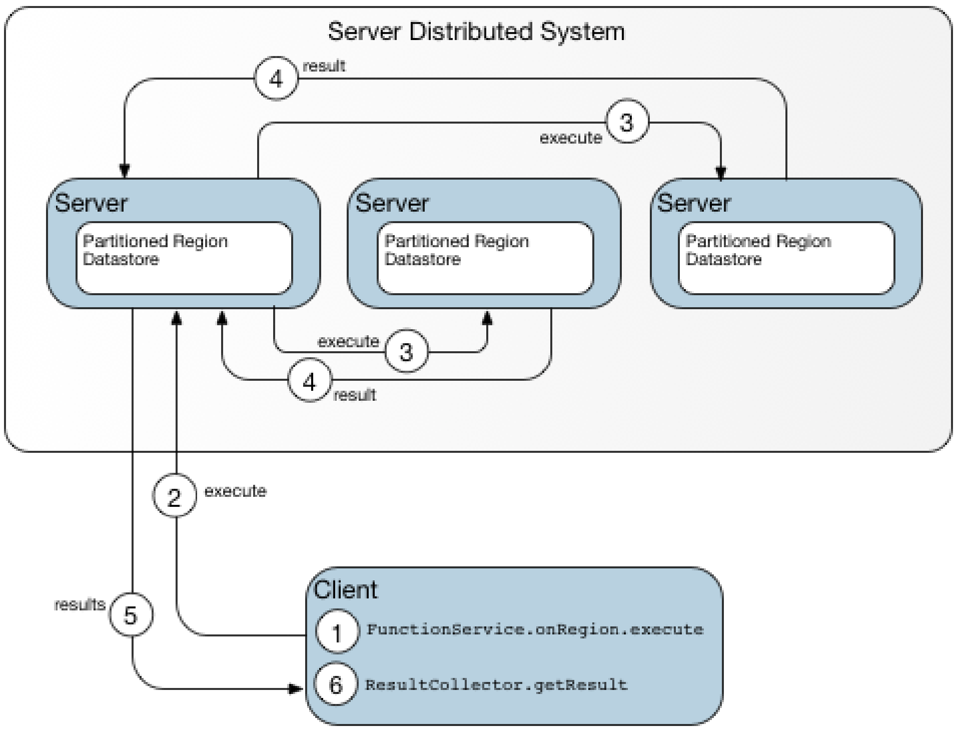
This figure shows the sequence of events for a data-independent function executed against members in a peer-to-peer cluster.



You can think of `onMembers()` as the peer-to-peer counterpart of a client-server call to `onServers()`. Because it is called from a peer of other members in the cluster, an `onMembers()` function invocation has access to detailed metadata and does not require the services of a locator. The caller invokes the function on itself, if appropriate, as well as other members in the cluster and collects the results of all of the function executions.

Data-dependent Function on a Region shows a data-dependent function run on a region.

Figure: Data-dependent Function on a Region

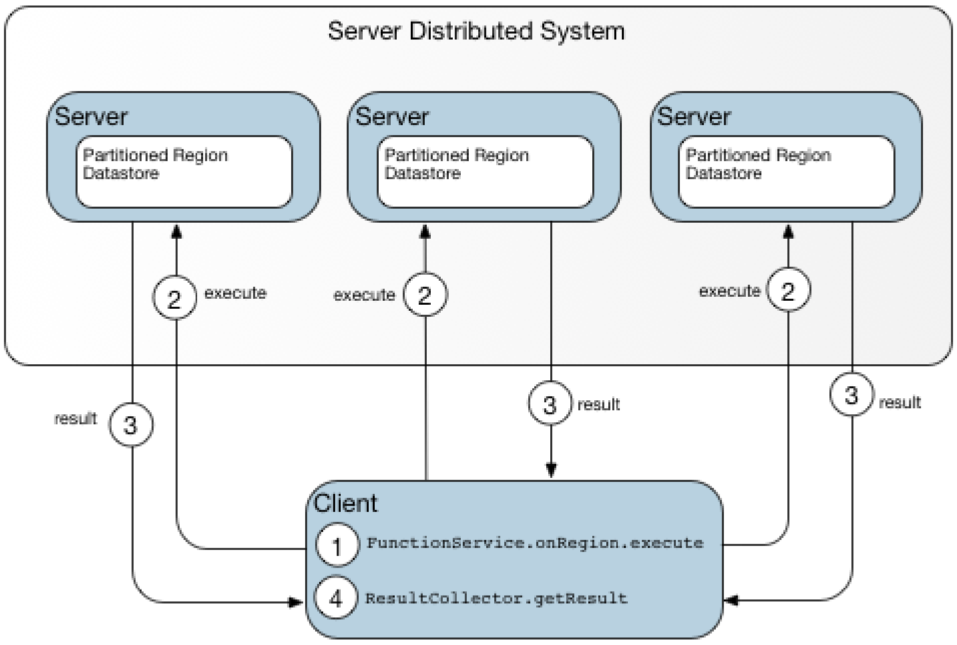

Server Distributed System
4 result
3 execute
Server
Partitioned Region Datastore
Server
Partitioned Region Datastore
Server
Partitioned Region Datastore
execute 3
4 result
2 execute
results 5
Client
1 FunctionService.onRegion.execute
6 ResultCollector.getResult

An `onRegion()` call requires more detailed metadata than a locator provides in its host:port identifier. This diagram shows the path followed when the client lacks detailed metadata regarding target locations, as on the first call or when previously obtained metadata is no longer up to date.

The first time a client invokes a function to be executed on a particular region of a cluster, the client's knowledge of target locations is limited to the host and port information provided by the locator. Given only this limited information, the client sends its execution request to whichever server is next in line to be called according to the pool allocation algorithm. Because it is a participant in the cluster, that server has access to detailed metadata and can dispatch the function call to the appropriate target locations. When the server returns results to the client, it sets a flag indicating whether a request to a different server would have provided a more direct path to the intended target. To improve efficiency, the client requests a copy of the metadata. With additional details regarding the bucket layout for the region, the client can act as its own dispatcher on subsequent calls and identify multiple targets for itself, eliminating at least one hop.

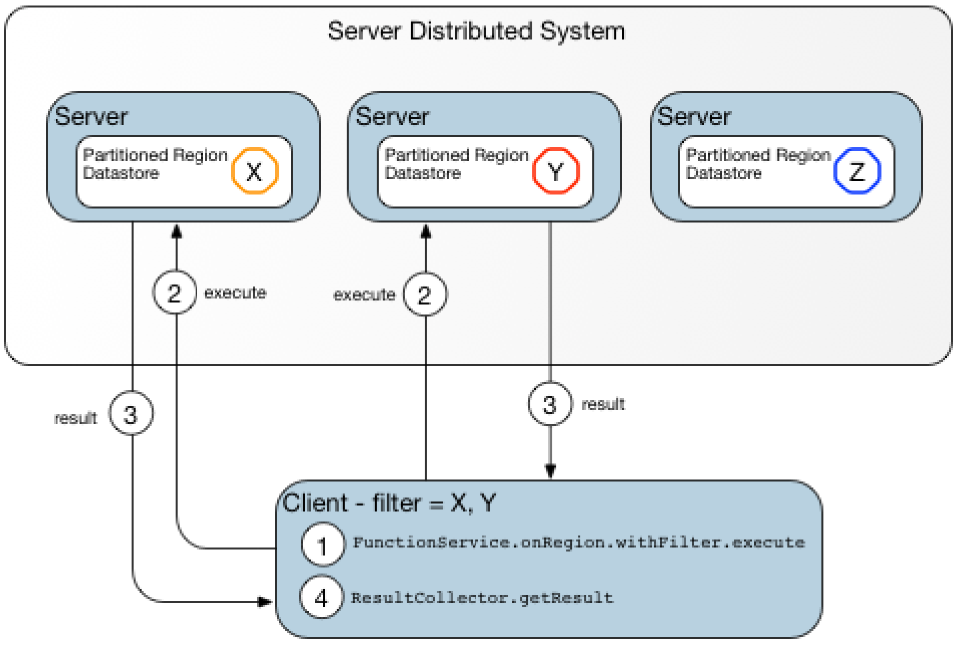
After it has obtained current metadata, the client can act as its own dispatcher on subsequent calls, identifying multiple targets for itself and eliminating one hop, as shown in Data-dependent function after obtaining current metadata.

Figure: Data-dependent function after obtaining current metadata


Server Distributed System
Server
Partitioned Region Datastore
Server
Partitioned Region Datastore
Server
Partitioned Region Datastore
2 execute
execute 2
execute 2
result 3
3 result
3 result
Client
1 FunctionService.onRegion.execute
4 ResultCollector.getResult

Data-dependent Function on a Region with Keys shows the same data-dependent function with the added specification of a set of keys on which to run.

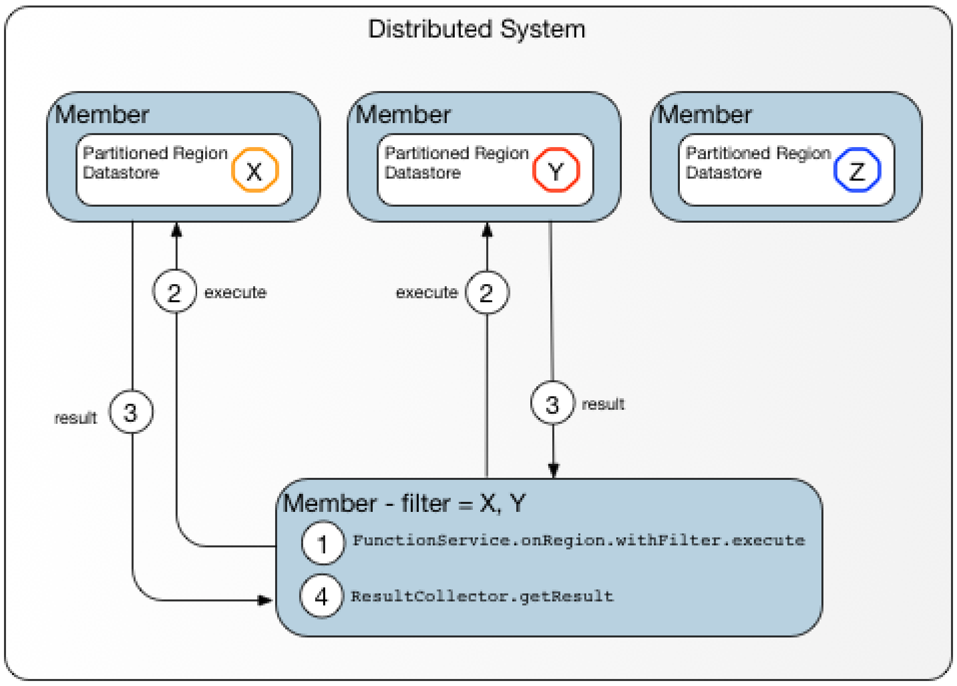
Figure: Data-dependent Function on a Region with Keys


Server Distributed System
Server
Partitioned Region Datastore
X
Server
Partitioned Region Datastore
Y
Server
Partitioned Region Datastore
Z
2 execute
execute 2
result 3
3 result
Client - filter = X, Y
1 FunctionService.onRegion.withFilter.execute
4 ResultCollector.getResult

Servers that do not hold any keys are left out of the function execution.

Peer-to-peer Data-dependent Function shows a peer-to-peer data-dependent call.

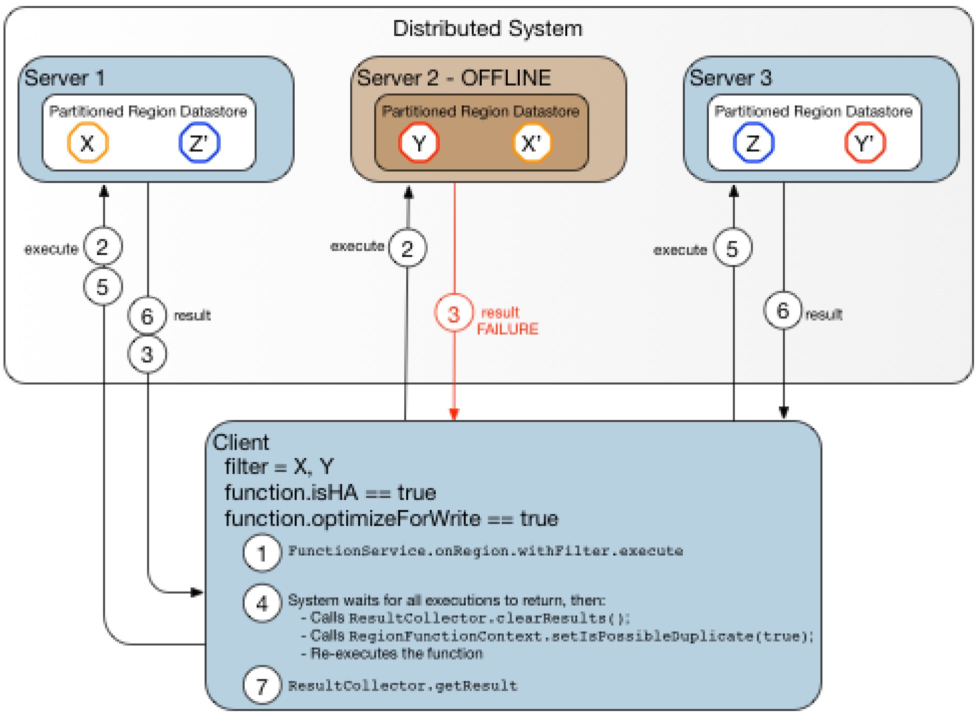
Figure: Peer-to-peer Data-dependent Function


Distributed System
Member
Partitioned Region Datastore
X
Member
Partitioned Region Datastore
Y
Member
Partitioned Region Datastore
Z
2 execute
execute 2
result 3
3 result
Member - filter = X, Y
1 FunctionService.onRegion.withFilter.execute
4 ResultCollector.getResult

The caller is a member of the cluster, not an external client, so the function runs in the caller's cluster. Note the similarities between this diagram and the preceding figure (Data-dependent Function on a Region with Keys), which shows a client-server model where the client has up-to-date metadata regarding target locations within the cluster.

Client-server system with Up-to-date Target Metadata demonstrates a sequence of steps in a call to a highly available function in a client-server system in which the client has up-to-date metadata regarding target locations.

Figure: Client-server system with Up-to-date Target Metadata


Distributed System
Server 1
Partitioned Region Datastore
X
Z'
Server 2 - OFFLINE
Partitioned Region Datastore
Y
X'
Server 3
Partitioned Region Datastore
Z
Y'
execute 2
5
6 result
3
execute 2
3 result FAILURE
execute 5
6 result
Client
filter = X, Y
function.isHA == true
function.optimizeForWrite == true
1 FunctionService.onRegion.withFilter.execute
4 System waits for all executions to return, then:
- Calls ResultCollector.clearResults();
- Calls RegionFunctionContext.setIsPossibleDuplicate(true);
- Re-executes the function
7 ResultCollector.getResult

In this example, three primary keys (X, Y, Z) and their secondary copies (X', Y', Z') are distributed among three servers. Because `optimizeForWrite` is `true`, the system first attempts to invoke the function where the primary keys reside: Server 1 and Server 2. Suppose, however, that Server 2 is off-line for some reason, so the call targeted for key Y fails. Because `isHA` is set to `true`, the call is retried on Server 1 (which succeeded the first time, so likely will do so again) and Server 3, where key Y' resides. This time, the function call returns successfully. Calls to highly available functions retry until they obtain a successful result or they reach a retry limit.

Executing a Function in VMware GemFire

In this procedure it is assumed that you have your members and regions defined where you want to run functions.

Main tasks:

1. Write the function code.
2. Register the function on all servers where you want to execute the function. The easiest way to register a function is to use the `gfsh deploy` command to deploy the JAR file containing the function code. Deploying the JAR automatically registers the function for you. See Register the Function Automatically by Deploying a JAR for details. Alternatively, you can write the XML or application code to register the function. See Register the Function Programmatically for details.
3. Write the application code to run the function and, if the function returns results, to handle the results.
4. If your function returns results and you need special results handling, code a custom `ResultsCollector` implementation and use it in your function execution.

Write the Function Code

To write the function code, you implement the `Function` interface in the `org.apache.geode.cache.execute` package.

Code the methods you need for the function. These steps do not have to be done in this order.

- Implement `getId` to return a unique name for your function. You can use this name to access the function through the `FunctionService` API.
- For high availability:
    1. Code `isHa` to return true to indicate to VMware GemFire that it can re-execute your function after one or more members fails
    2. Code your function to return a result
    3. Code `hasResult` to return true
- Code `hasResult` to return true if your function returns results to be processed and false if your function does not return any data - the fire and forget function.
- If the function will be executed on a region, implement `optimizeForWrite` to return false if your function only reads from the cache, and true if your function updates the cache. The

method only works if, when you are running the function, the `Execution` object is obtained through a `FunctionService onRegion` call. `optimizeForWrite` returns false by default.

- If the function should be run with an authorization level other than the default of `DATA:WRITE`, implement an override of the `Function.getRequiredPermissions()` method. See Authorization of Function Execution for details on this method.
- Code the `execute` method to perform the work of the function.
  1. Make `execute` thread safe to accommodate simultaneous invocations.
  2. For high availability, code `execute` to accommodate multiple identical calls to the function. Use the `RegionFunctionContext isPossibleDuplicate` to determine whether the call may be a high-availability re-execution. This boolean is set to true on execution failure and is false otherwise. **Note:** The `isPossibleDuplicate` boolean can be set following a failure from another member's execution of the function, so it only indicates that the execution might be a repeat run in the current member.
  3. Use the function context to get information about the execution and the data:
     - The context holds the function ID, the `ResultSender` object for passing results back to the originator, and function arguments provided by the member where the function originated.
     - The context provided to the function is the `FunctionContext`, which is automatically extended to `RegionFunctionContext` if you get the `Execution` object through a `FunctionService onRegion` call.
     - For data dependent functions, the `RegionFunctionContext` holds the `Region` object, the `Set` of key filters, and a boolean indicating multiple identical calls to the function, for high availability implementations.
     - For partitioned regions, the `PartitionRegionHelper` provides access to additional information and data for the region. For single regions, use `getLocalDataForContext`. For colocated regions, use `getLocalColocatedRegions`. **Note:** When you use `PartitionRegionHelper.getLocalDataForContext`, `putIfAbsent` may not return expected results if you are working on local data set instead of the region.
  4. To propagate an error condition or exception back to the caller of the function, throw a FunctionException from the `execute` method. VMware GemFire transmits the exception back to the caller as if it had been thrown on the calling side. See the Java API documentation for FunctionException for more information.

Example function code:

```
import java.io.Serializable;
import java.util.HashSet;
import java.util.Iterator;
import java.util.Set;

import org.apache.geode.cache.execute.Function;
import org.apache.geode.cache.execute.FunctionContext;
import org.apache.geode.cache.execute.FunctionException;
import org.apache.geode.cache.execute.RegionFunctionContext;
import org.apache.geode.cache.partition.PartitionRegionHelper;
```

```
public class MultiGetFunction implements Function {

  public void execute(FunctionContext fc) {
    if(! (fc instanceof RegionFunctionContext)){
      throw new FunctionException("This is a data aware function, and has
 to be called using FunctionService.onRegion.");
    }
    RegionFunctionContext context = (RegionFunctionContext)fc;
    Set keys = context.getFilter();
    Set keysTillSecondLast = new HashSet();
    int setSize = keys.size();
    Iterator keysIterator = keys.iterator();
    for(int i = 0; i < (setSize -1); i++)
    {
      keysTillSecondLast.add(keysIterator.next());
    }
    for (Object k : keysTillSecondLast) {
      context.getResultSender().sendResult(
          (Serializable)PartitionRegionHelper.getLocalDataForContext(context)
              .get(k));
    }
    Object lastResult = keysIterator.next();
    context.getResultSender().lastResult(
        (Serializable)PartitionRegionHelper.getLocalDataForContext(context)
            .get(lastResult));
  }

  public String getId() {
    return getClass().getName();
  }
}
```

Register the Function Automatically by Deploying a JAR

When you deploy a JAR file that contains a Function (in other words, contains a class that implements the Function interface), the Function will be automatically registered via the `FunctionService.registerFunction` method.

To register a function by using `gfsh`:

1. Package your class files into a JAR file.
2. Start a `gfsh` prompt. If necessary, start a locator and connect to the cluster where you want to run the function.
3. At the gfsh prompt, type the following command:

   ```
   gfsh>deploy --jar=group1_functions.jar
   ```

   where group1_functions.jar corresponds to the JAR file that you created in step 1.

If another JAR file is deployed (either with the same JAR filename or another filename) with the same Function, the new implementation of the Function will be registered, overwriting the old one. If a JAR file is undeployed, any Functions that were auto-registered at the time of deployment will be unregistered. Since deploying a JAR file that has the same name multiple times results in the JAR being un-deployed and re-deployed, Functions in the JAR will be unregistered and re-

registered each time this occurs. If a Function with the same ID is registered from multiple differently named JAR files, the Function will be unregistered if either of those JAR files is re-deployed or un-deployed.

See Deploying Application JARs to VMware GemFire Members for more details on deploying JAR files.

Register the Function Programmatically

This section applies to functions that are invoked using the `Execution.execute(String functionId)` signature. When this method is invoked, the calling application sends the function ID to all members where the `Function.execute` is to be run. Receiving members use the ID to look up the function in the local `FunctionService`. In order to do the lookup, all of the receiving member must have previously registered the function with the function service.

The alternative to this is the `Execution.execute(Function function)` signature. When this method is invoked, the calling application serializes the instance of `Function` and sends it to all members where the `Function.execute` is to be run. Receiving members deserialize the `Function` instance, create a new local instance of it, and run execute from that. This option is not available for non-Java client invocation of functions on servers.

Your Java servers must register functions that are invoked by non-Java clients. You may want to use registration in other cases to avoid the overhead of sending `Function` instances between members.

Register your function using one of these methods:

- XML:

  ```
  <cache>
      ...
      </region>
  <function-service>
    <function>
      <class-name>com.bigFatCompany.tradeService.cache.func.TradeCalc</class-name
  >
    </function>
  </function-service>
  ```

- Java:

  ```
  myFunction myFun = new myFunction();
  FunctionService.registerFunction(myFun);
  ```

  Note: Modifying a function instance after registration has no effect on the registered function. If you want to execute a new function, you must register it with a different identifier.

Run the Function

This assumes you've already followed the steps for writing and registering the function.

In every member where you want to explicitly execute the function and process the results, you can use the `gfsh` command line to run the function or you can write an application to run the function.

Running the Function Using gfsh

1. Start a gfsh prompt.
2. If necessary, start a locator and connect to the cluster where you want to run the function.
3. At the gfsh prompt, type the following command:

   ```
   gfsh> execute function --id=function_id
   ```

   Where *function_id* equals the unique ID assigned to the function. You can obtain this ID using the `Function.getId` method.

See Function Execution Commands for more `gfsh` commands related to functions.

Running the Function via API Calls

1. Use one of the `FunctionService on*` methods to create an `Execute` object. The `on*` methods, `onRegion`, `onMembers`, etc., define the highest level where the function is run. For colocated partitioned regions, use `onRegion` and specify any one of the colocated regions. The function run using `onRegion` is referred to as a data dependent function - the others as data-independent functions.
2. Use the `Execution` object as needed for additional function configuration. You can:
   - Provide a key `Set` to `withFilters` to narrow the execution scope for `onRegion` `Execution` objects. You can retrieve the key set in your `Function execute` method through `RegionFunctionContext.getFilter`.
   - Provide function arguments to `setArguments`. You can retrieve these in your `Function execute` method through `FunctionContext.getArguments`.
   - Define a custom `ResultCollector`
3. Call the `Execution` object to `execute` method to run the function.
4. If the function returns results, call `getResult` from the results collector returned from `execute` and code your application to do whatever it needs to do with the results. **Note:** For high availability, you must call the `getResult` method.

Example of running the function - for executing members:

```
MultiGetFunction function = new MultiGetFunction();
FunctionService.registerFunction(function);

writeToStdout("Press Enter to continue.");
stdinReader.readLine();

Set keysForGet = new HashSet();
keysForGet.add("KEY_4");
keysForGet.add("KEY_9");
keysForGet.add("KEY_7");

Execution execution = FunctionService.onRegion(exampleRegion)
    .withFilter(keysForGet)
    .setArguments(Boolean.TRUE)
    .withCollector(new MyArrayListResultCollector());
```

```
ResultCollector rc = execution.execute(function);
// Retrieve results, if the function returns results
List result = (List)rc.getResult();
```

Write a Custom Results Collector

This topic applies to functions that return results.

When you execute a function that returns results, the function stores the results into a `ResultCollector` and returns the `ResultCollector` object. The calling application can then retrieve the results through the `ResultCollector` `getResult` method. Example:

```
ResultCollector rc = execution.execute(function);
List result = (List)rc.getResult();
```

VMware GemFire's default `ResultCollector` collects all results into an `ArrayList`. Its `getResult` methods block until all results are received. Then they return the full result set.

To customize results collecting:

1. Write a class that extends `ResultCollector` and code the methods to store and retrieve the results as you need. Note that the methods are of two types:
    1. `addResult` and `endResults` are called by VMware GemFire when results arrive from the `Function` instance `SendResults` methods
    2. `getResult` is available to your executing application (the one that calls `Execution.execute`) to retrieve the results
2. Use high availability for `onRegion` functions that have been coded for it:
    1. Code the `ResultCollector` `clearResults` method to remove any partial results data. This readies the instance for a clean function re-execution.
    2. When you invoke the function, call the result collector `getResult` method. This enables the high availability functionality.
3. In your member that calls the function execution, create the `Execution` object using the `withCollector` method, and passing it your custom collector. Example:

    ```
    Execution execution = FunctionService.onRegion(exampleRegion)
        .withFilter(keysForGet)
        .setArguments(Boolean.TRUE)
        .withCollector(new MyArrayListResultCollector());
    ```

Targeting Single Members of a Member Group or Entire Member Groups

To execute a data independent function on a group of members or one member in a group of members, you can write your own nested function. You will need to write one nested function if you are executing the function from client to server and another nested function if you are executing a function from server to all members.

Developing REST Applications for VMware GemFire

Developing REST Applications for VMware GemFire provides background and instructions on how to program REST applications with VMware GemFire. VMware GemFire REST APIs allow you to access region data, queries and functions in your VMware GemFire deployment in wide variety of programming languages.

Note: This documentation covers the **v1** release of VMware GemFire REST APIs for developing applications.

- **VMware GemFire REST API Overview**

  By using the VMware GemFire REST application interface, you can immediately access VMware GemFire's data management capabilities in languages other than the natively supported Java language.

- **Prerequisites and Limitations for Writing REST Applications**

  Before development, understand the prerequisites and limitations of the current REST implementation in VMware GemFire.

- **Setup and Configuration**

  The VMware GemFire developer REST interface runs as an embedded HTTP or HTTPS service (Jetty server) within a VMware GemFire data node.

- **Using the Swagger UI to Browse REST APIs**

  VMware GemFire Developer REST APIs are integrated with the Swagger™ framework. This framework provides a browser-based test client that allows you to visualize and try out VMware GemFire REST APIs.

- **Developing REST Applications**

  This section provides guidelines on writing REST client applications for VMware GemFire.

- **Sample REST Applications**

  This section provides examples that illustrate how multiple clients, both REST and native, can access the same VMware GemFire region data.

- **Troubleshooting and FAQ**

  This section provides troubleshooting guidance and frequently asked questions about VMware GemFire Developer REST APIs.

- **VMware GemFire REST API Reference**

  This section summarizes all available VMware GemFire REST API resources and endpoints.

VMware GemFire REST API Overview

By using the VMware GemFire REST application interface, you can immediately access VMware GemFire's data management capabilities in languages other than the natively supported Java language.

You can write REST-enabled client applications for VMware GemFire in a variety of languages that use the open and standard HTTP protocol—for example, Ruby, Python, JavaScript and Scala—as well as already supported languages such as Java.

When you access VMware GemFire through the REST interface, objects are stored in VMware GemFire as PdxInstances. A PdxInstance is a light-weight wrapper around PDX serialized bytes. It provides applications with run-time access to fields of a PDX serialized object. This interoperable format allows your Java applications to operate on the same data as your REST applications.

As an added benefit, because VMware GemFire's REST interface stores objects as PdxInstances, you do not need to write corresponding Java classes to translate JSON data (which you must do with other REST interface providers such as Oracle Coherence). For example, consider the use case where a non-Java REST client application (Python, Ruby or Scala) performs VMware GemFire region operations with JSON data that represents employee data. Since the object is stored in VMware GemFire as a PdxInstance that can be automatically mapped to JSON, the user does not need to write a corresponding Employee.java class and also does not need to worry about related issues such as keeping the Employee object in the CLASSPATH.

See VMware GemFire PDX Serialization for more information on PDX serialization.

Prerequisites and Limitations for Writing REST Applications

Before development, it is important to understand the prerequisites and limitations of the VMware GemFire REST implementation.

VMware GemFire and REST-enabled applications accessing VMware GemFire are subject to the following rules and limitations:

- All domain objects, functions and function-arg classes must be properly configured and registered in the VMware GemFire deployment. Any functions that you wish to execute through the REST API must be available on the target member's CLASSPATH.
- The current implementation supports only the **application/json** MIME type. Other return types (XML, objects, and so on) are not supported. Plain text is supported as a return type for some error messages.
- Keys are strictly of type String. For example, the request `PUT http://localhost:8080/geode/v1/customers/123.456` will add an entry for key ("123.456") of type String.
- Some special formats of JSON documents are not supported in VMware GemFire REST. See Key Types and JSON Support for examples.
- To achieve interoperability between VMware GemFire Java clients (or VMware GemFire native clients) and REST clients, the following rules must be followed:

  - All VMware GemFire Java and native client classes operating on data also accessed by the REST interface must be PDX serialized, either via PDX autoserialization or by implementing `PdxSerializable`.
  - VMware GemFire Java clients and native clients can retrieve REST-enabled data either as a `PdxInstance` or as an actual object by using the `PdxInstance.getObject` method. If you use the latter method, you must first declare the object type (@type) in your POST or PUT request payload when creating the object in REST; and secondly, the Java client must have the actual domain class in its CLASSPATH.
- Objects returned by REST-invoked functions must be returned as PdxInstance objects or other data types that can be written to JSON. You cannot return Java objects.
- REST client applications do not support single hop access or notification features.
- Specifying subregions as endpoints is not supported.

Setup and Configuration

The VMware GemFire Developer REST interface runs as an embedded HTTP or HTTPS service (Jetty server) within one or more VMware GemFire servers.

REST API Libraries

All VMware GemFire REST interface classes and required JAR files are distributed as a WAR file with the VMware GemFire product distribution. You can find the file in the following location:

`install-dir/tools/Extensions/geode-web-api-n.n.n.war`

where *install-dir* is the server installation directory and *n.n.n* is a version number.

Setting a `GEODE_HOME` environment variable with a path to the VMware GemFire installation directory allows a server launcher to find the WAR file without any changes to the CLASSPATH.

Enabling the REST API

The REST API service for application development runs only on servers; you cannot run the service on a locator.

To enable the Developer REST API service on a given server, use the `gfsh start server` command with the `--start-rest-api` option, or set the `start-dev-rest-api` property to `true` for the server via the ServerLauncher API. This starts an embedded Jetty server and deploys the Developer REST API WAR file on that server.

Enabling the REST API on Multiple Servers

You can configure multiple REST-enabled servers in a single cluster. Each server should have a separate host name and unique end point. To ensure that the server is reachable on a machine with multiple NIC addresses, use `http-service-bind-address` to bind an address to the REST API service (as well as the other embedded web services, such as Pulse).

You can configure the Developer REST API service to run over HTTPS by enabling SSL for the `http` component in `gemfire.properties` or `gfsecurity.properties`, or on server startup. See SSL for details on configuring SSL parameters. These SSL parameters apply to all HTTP services hosted on the configured server, which can include the following:

- Developer REST API service
- Management REST API service (for remote cluster management)
- Pulse monitoring tool

Starting the REST API Service

To start a REST API service-enabled VMware GemFire deployment, configure PDX serialization for your cluster, then start the service on one or more server nodes.

Configure PDX for your cluster

You must configure PDX if either or both of the following conditions apply:

- Application peer member caches will access REST-accessible regions (resources) with `Region.get(key)`.
- Your deployment has persistent regions that must be available as resources to the REST API.

To configure PDX in your cluster, perform the following steps:

1. Start a locator running the cluster configuration service (enabled by default). For example:

   ```
   gfsh>start locator --name=locator1
   ```

2. If your deployment has application peer member caches (for example, Java clients) that must also access REST-accessible Regions (resources), use the following gfsh command:

   ```
   gfsh>configure pdx --read-serialized=true
   ```

   Note: You do not need to configure `--read-serialized=true` if no application peer member caches are accessing the REST-accessible regions (resources) in your deployment.

3. If your deployment contains **persistent regions** that must be REST-accessible, use the following gfsh command:

   ```
   gfsh>configure pdx --disk-store
   ```

   This command sets `pdx persistent` equal to true and sets the disk-store-name to DEFAULT. If desired, specify an existing disk store name as the value for `--disk-store`.

4. If both of the above cases apply to your deployment, then configure PDX with the following single command:

   ```
   gfsh>configure pdx --read-serialized=true --disk-store
   ```

   After you have configured PDX for your caches, then proceed with starting your REST-enabled servers and other servers.

Start the REST API Service on One or More Servers

As described above, you can start the REST API service on a server by using `gfsh start server --start-rest-api`, or by setting the VMware GemFire property `start-dev-rest-api` to `true`. If you wish to start the service on multiple servers, use `http-service-bind-address` and `http-service-port` to identify the cache server and specific port that will host REST services. If you do not specify the `http-service-port`, the default port is 7070, which may collide with other locators and servers. If you do not specify `http-service-bind-address`, the HTTP service will bind to all local addresses by default.

Note: If your application will be running in a VM (as when running in the cloud, for example), it is good practice to specify `http-service-bind-address` and `http-service-port` so they will be publicly visible. The default values may not be visible outside the VM in which the application is running.

For example:

```
gfsh>start server --name=server1 --start-rest-api=true \
--http-service-port=8080 --http-service-bind-address=localhost
```

Any server that hosts data, even a server acting as a JMX manager, can start the Developer REST API service. For example, to start the service on a server that is also a JMX manager, you would run:

```
gfsh>start server --name=server1  --start-rest-api=true \
--http-service-port=8080 --http-service-bind-address=localhost \
--J=-Dgemfire.jmx-manager=true --J=-Dgemfire.jmx-manager-start=true
```

Note that when started as a JMX Manager, the server will also host the Pulse web application in the same HTTP service.

You may need to specify a CLASSPATH to load any functions that need to be made available to your REST services. For example:

```
gfsh>start server --name=server1 --start-rest-api=true \
--http-service-port=8080 --http-service-bind-address=localhost \
--classpath=/myapps/testfunctions.jar
```

You can specify these properties either upon server startup or in the server’s gemfire.properties configuration file. For example:

```
gfsh>start server --name=serverX --server-port=40405 --cache-xml-file=cache-config.xml
\
--properties-file=gemfire.properties --classpath=/myapps/testfunctions.jar
```

where gemfire.properties contains:

```
http-service-port=8080
http-service-bind-address=localhost
start-dev-rest-api=true
```

Verify That The Service is Running

Verify that the VMware GemFire REST API service is up and running. To validate this, you can perform the following checks:

1. Test the list resources endpoint (this step assumes that you have regions defined on your cluster):

   ```
   curl -i http://localhost:8080/geode/v1
   ```

2. Examine the server logs for the following messages:

   ```
   [info 2017/06/13 13:48:14.090 PDT gfsec-server1 <main> tid=0x1] Initializing Sp
   ring FrameworkServlet 'geode-mgmt'
   [info 2017/06/13 13:48:14.091 PDT gfsec-server1 <main> tid=0x1] FrameworkServle
   t 'geode-mgmt': initialization started
   ```

3. Open a browser and enter the following URL to browse the Swagger-enabled REST APIs:

   ```
   http://<http-service-bind-address>:<http-service-port>/geode/docs/index.html
   ```

   where *http-service-bind-address* is the address and *http-service-port* is the port number that you specified when starting the Development REST API service on the server. For example, based on the server started in an earlier example, you would enter:

   ```
   http://localhost:8080/geode/docs/index.html
   ```

If you did not specify these properties upon server startup or in `gemfire.properties`, then use the default of localhost and port 7070. See Using the Swagger UI to Browse REST APIs for more information.

Implementing Authentication

To turn on integrated security, start your servers and locators with the security-manager property set in your gemfire.properties file or on the gfsh command-line. The following example uses the sample implementation that is included in the VMware GemFire source, `org.apache.geode.examples.security.ExampleSecurityManager`.

This implementation requires a JSON security configuration file which defines the allowed users and their corresponding permissions. (See the javadocs for `ExampleSecurityManager` for details on how to compose the JSON file.) Place a copy of the JSON security configuration file in the execution directory of each security-enabled member, then specify `--classpath=.` in the start command for each of those members.

To start a server using a username and password that are defined in that server's security configuration, include the `--user=username` and `--password=password` options in the server's start command:

For example, suppose the JSON config file defines user "super-user" with password "1234567":

```
gfsh>start server --name=server1 --start-rest-api=true \
--http-service-port=8080 --http-service-bind-address=localhost \
--J=-Dgemfire.security-manager=org.apache.geode.examples.security.ExampleSecurityManag
```

```
er \
--classpath=. --user=super-user --password=1234567
```

To contact the server through the REST interface, you must provide the username and password. Various REST GUI interfaces provide different ways of accomplishing this. The `curl` command offers the `--user` (or `-u`) option for this purpose, where username and password are specified as a colon-separated pair:

```
curl -i --user super-user:1234567 http://localhost:8080/geode/v1
```

In a simple URL, such as in a browser address bar, the credentials can be given as a prefix to the host name in the form `username:password@`:

```
http://super-user:1234567@localhost:8080/geode/v1
```

Programmatic Startup

You can also start and configure VMware GemFire REST services programmatically. For example:

```
import org.apache.geode.distributed.ServerLauncher;

public class MyEmbeddedRestServer {

public static void main(String[] args){
     ServerLauncher serverLauncher  = new ServerLauncher.Builder()
       .set("start-dev-rest-api", "true")
       .set("http-service-port", "8080")
       .set("http-service-bind-address", "localhost")
       .setPdxReadSerialized(true)
       .build();

      serverLauncher.start();

      System.out.println("REST server successfully started");
    }
}
```

You can then verify that the Developer REST API service has been started programmatically by visiting the following URL:

```
http://localhost:8080/geode/docs/index.html
```

Using the Swagger UI to Browse REST APIs

VMware GemFire Developer REST APIs are integrated with the Swagger™ framework. This framework provides a browser-based test client that allows you to visualize and try out VMware GemFire REST APIs.

Swagger application JARs are included in the VMware GemFire REST application WAR; you do not need to install any additional libraries to use Swagger.

The following example demonstrates how to access the Swagger UI to browse the APIs.

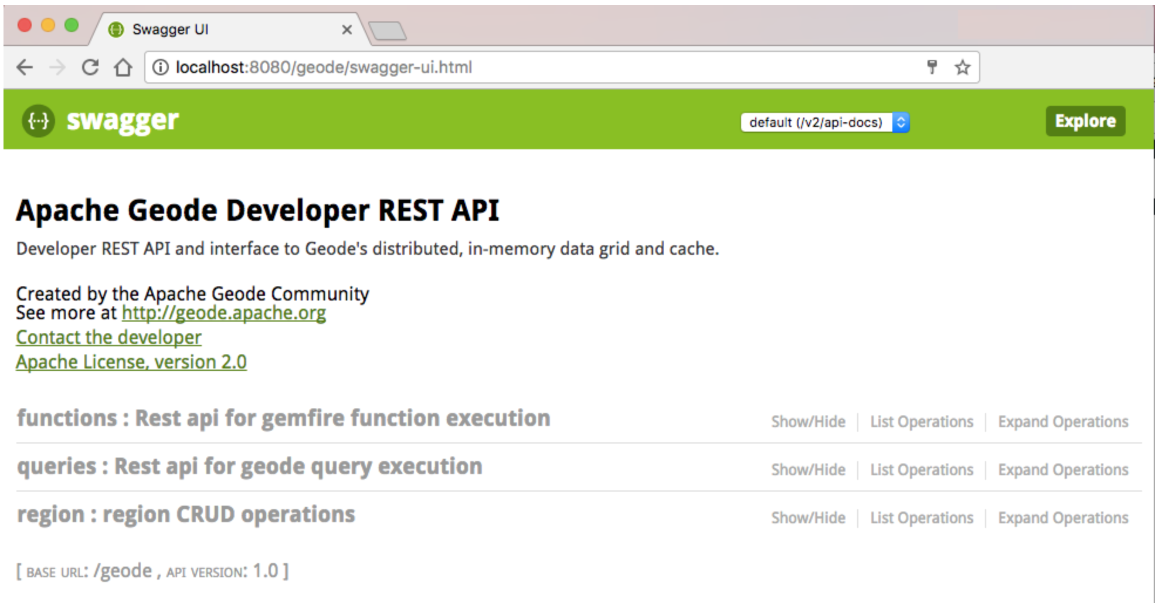
1. Start a VMware GemFire Locator and a Developer REST API-enabled server as described in Setup and Configuration. Specify an `http-service-port` for the developer REST service, as the default port, 7070, is already taken by the locator. For example:

   ```
   gfsh>start locator --name=locator1
   Starting a VMware GemFire Locator in /Users/admin/apache-geode-1.2.0/locator
   1...
   ....
   gfsh>start server --name=server1  --start-rest-api=true \
   --http-service-bind-address=localhost --J=-Dgemfire.http-service-port=8080
   ```

2. To access Swagger, open a browser and enter the following URL. For example:

   ```
   http://localhost:8080/geode/swagger-ui.html
   ```

   The following Web page appears:



Apache Geode Developer REST API

Developer REST API and interface to Geode's distributed, in-memory data grid and cache.

Created by the Apache Geode Community
See more at http://geode.apache.org
Contact the developer
Apache License, version 2.0

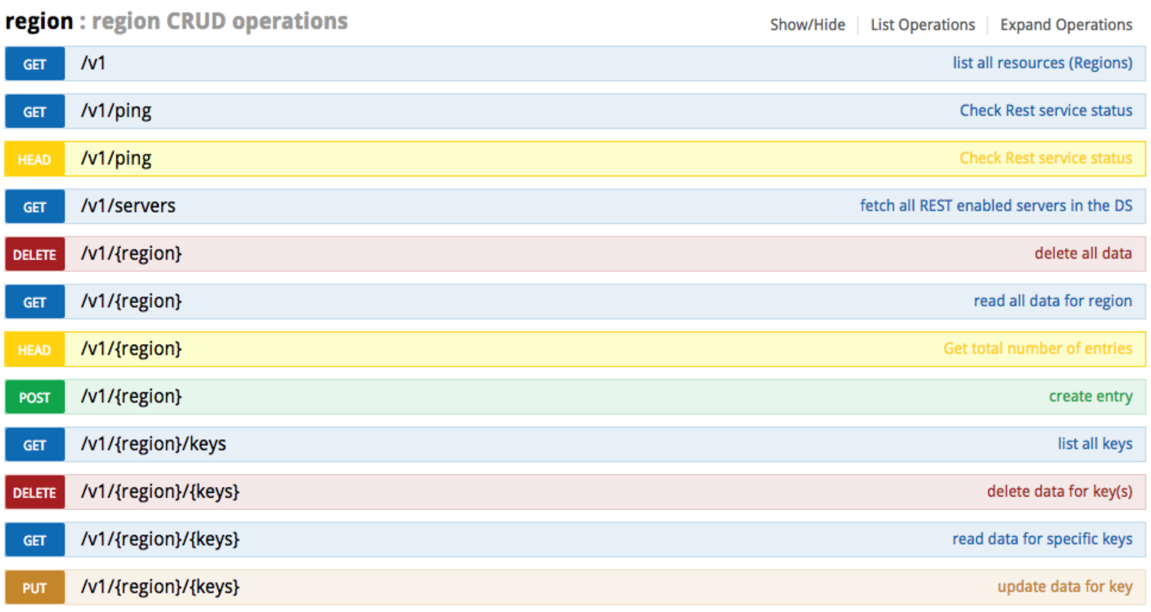
functions : Rest api for gemfire function execution — Show/Hide | List Operations | Expand Operations
queries : Rest api for geode query execution — Show/Hide | List Operations | Expand Operations
region : region CRUD operations — Show/Hide | List Operations | Expand Operations

[BASE URL: /geode , API VERSION: 1.0]

3. Using gfsh, create one or more regions on the REST API server. For example:

   ```
   gfsh>create region --name=region1 --type=REPLICATE --key-constraint=java.lang.S
   tring
   Member  | Status
   ------- | ------------------------------------------
   server1 | Region "/region1" created on "server1"
   ```

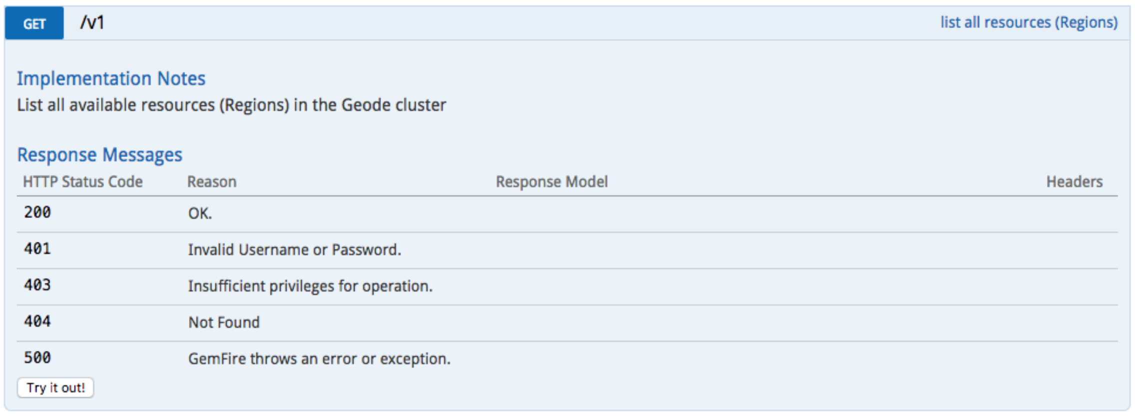
4. In Swagger, click on **region : region CRUD operations** to list all the available endpoints for accessing regions.



region : region CRUD operations — Show/Hide | List Operations | Expand Operations

Method	Endpoint	Description
GET	/v1	list all resources (Regions)
GET	/v1/ping	Check Rest service status
HEAD	/v1/ping	Check Rest service status
GET	/v1/servers	fetch all REST enabled servers in the DS
DELETE	/v1/{region}	delete all data
GET	/v1/{region}	read all data for region
HEAD	/v1/{region}	Get total number of entries
POST	/v1/{region}	create entry
GET	/v1/{region}/keys	list all keys
DELETE	/v1/{region}/{keys}	delete data for key(s)
GET	/v1/{region}/{keys}	read data for specific keys
PUT	/v1/{region}/{keys}	update data for key

5. In the list of **region** endpoints, click on the **GET /v1** endpoint link. The page displays additional request and response information about the API.



GET /v1 — list all resources (Regions)

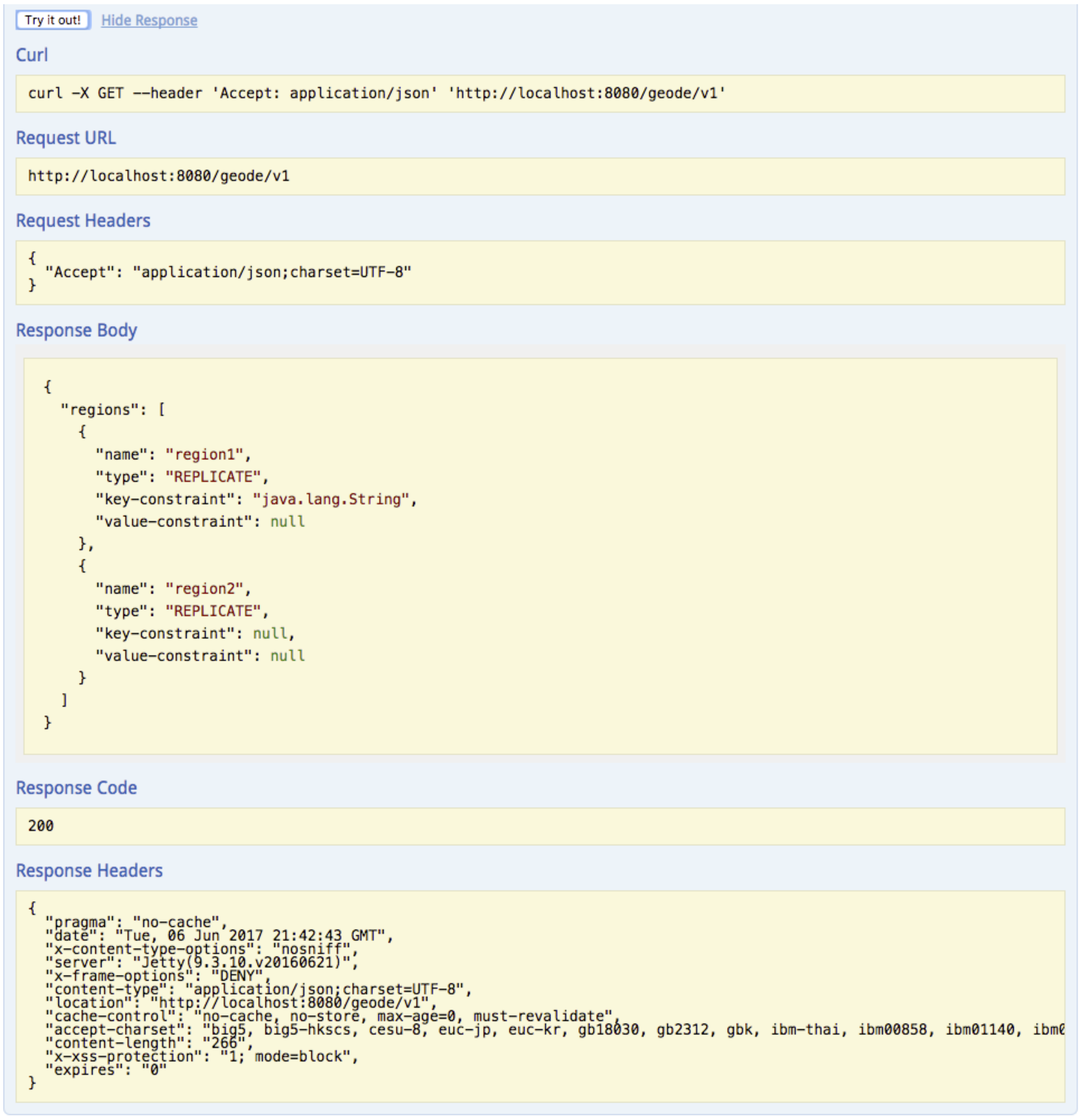
Implementation Notes

List all available resources (Regions) in the Geode cluster

Response Messages

HTTP Status Code	Reason	Response Model	Headers
200	OK.		
401	Invalid Username or Password.		
403	Insufficient privileges for operation.		
404	Not Found		
500	GemFire throws an error or exception.		

6. Click the **Try it out!** button. Any regions you added in step 5 are returned in the response body.



Curl

```
curl -X GET --header 'Accept: application/json' 'http://localhost:8080/geode/v1'
```

Request URL

```
http://localhost:8080/geode/v1
```

Request Headers

```
{
  "Accept": "application/json;charset=UTF-8"
}
```

Response Body

```
{
  "regions": [
    {
      "name": "region1",
      "type": "REPLICATE",
      "key-constraint": "java.lang.String",
      "value-constraint": null
    },
    {
      "name": "region2",
      "type": "REPLICATE",
      "key-constraint": null,
      "value-constraint": null
    }
  ]
}
```

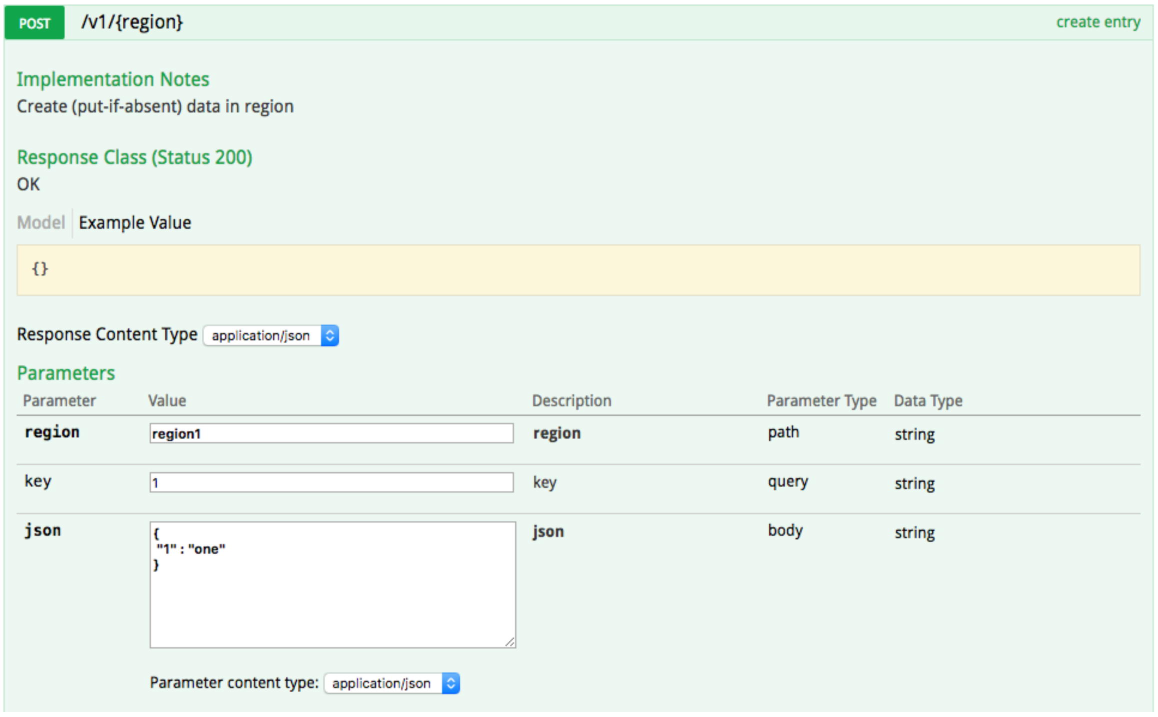
Response Code

```
200
```

Response Headers

```
{
  "pragma": "no-cache",
  "date": "Tue, 06 Jun 2017 21:42:43 GMT",
  "x-content-type-options": "nosniff",
  "server": "Jetty(9.3.10.v20160621)",
  "x-frame-options": "DENY",
  "content-type": "application/json;charset=UTF-8",
  "location": "http://localhost:8080/geode/v1",
  "cache-control": "no-cache, no-store, max-age=0, must-revalidate",
  "accept-charset": "big5, big5-hkscs, cesu-8, euc-jp, euc-kr, gb18030, gb2312, gbk, ibm-thai, ibm00858, ibm01140, ibm0
  "content-length": "266",
  "x-xss-protection": "1; mode=block",
  "expires": "0"
}
```

7. Add an entry to the region by expanding the **POST /v1/{region}** endpoint.



POST /v1/{region} — create entry

Implementation Notes

Create (put-if-absent) data in region

Response Class (Status 200)

OK

Model | Example Value

```
{}
```

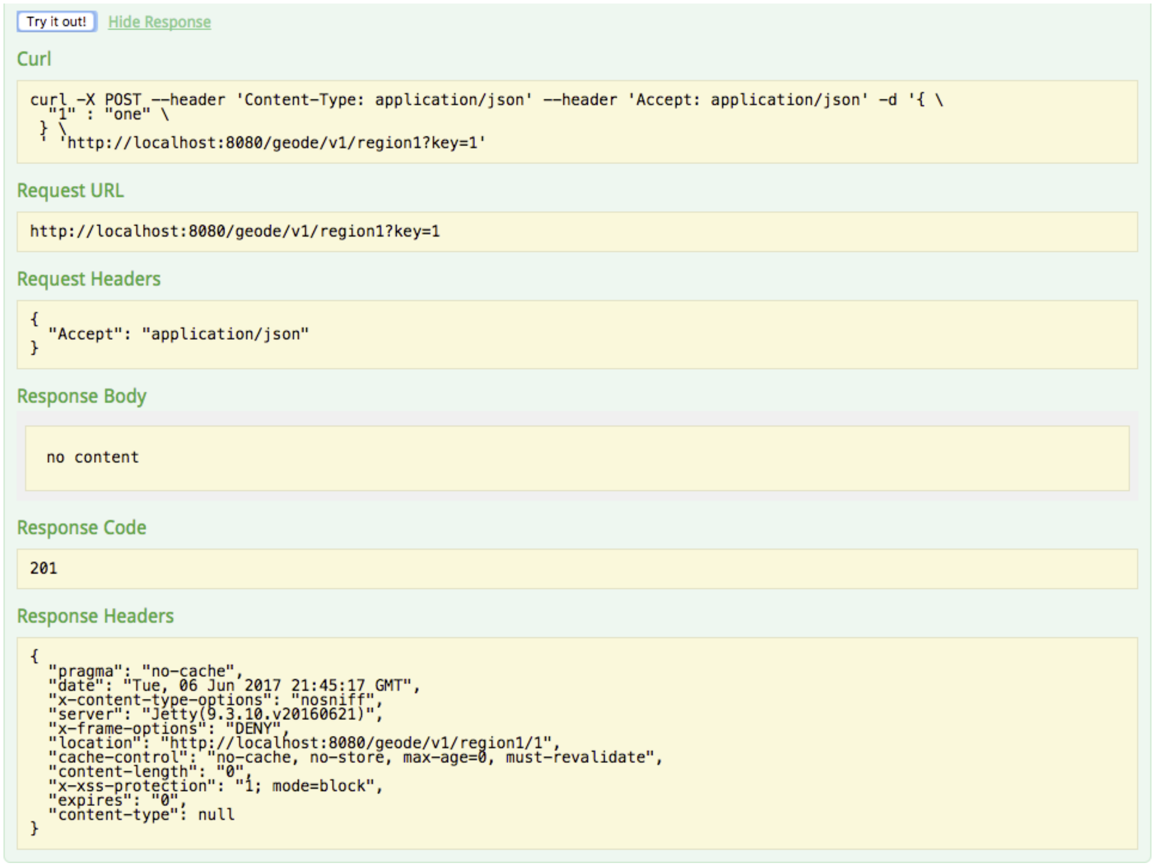
Response Content Type: application/json

Parameters

Parameter	Value	Description	Parameter Type	Data Type
region	region1	region	path	string
key	1	key	query	string
json	{ "1" : "one" }	json	body	string

Parameter content type: application/json

8. Click the **Try it out!** button to see the response body and response code.



Curl

```
curl -X POST --header 'Content-Type: application/json' --header 'Accept: application/json' -d '{ \
   "1" : "one" \
 } \
 ' 'http://localhost:8080/geode/v1/region1?key=1'
```

Request URL

```
http://localhost:8080/geode/v1/region1?key=1
```

Request Headers

```
{
  "Accept": "application/json"
}
```

Response Body

```
no content
```

Response Code

```
201
```

Response Headers

```
{
  "pragma": "no-cache",
  "date": "Tue, 06 Jun 2017 21:45:17 GMT",
  "x-content-type-options": "nosniff",
  "server": "Jetty(9.3.10.v20160621)",
  "x-frame-options": "DENY",
  "location": "http://localhost:8080/geode/v1/region1/1",
  "cache-control": "no-cache, no-store, max-age=0, must-revalidate",
  "content-length": "0",
  "x-xss-protection": "1; mode=block",
  "expires": "0",
  "content-type": null
}
```

You can use the Swagger interface to try out additional VMware GemFire API endpoints and view sample responses.

For more information on Swagger, see the Swagger website and the OpenAPI specification.

Developing REST Applications

This section provides guidelines on writing REST client applications for VMware GemFire.

You can browse, query, update and delete data stored in your VMware GemFire deployment. You can also manage and execute pre-deployed functions on VMware GemFire members.

- **Working with Regions**

  The VMware GemFire REST APIs provide basic CRUD (create, read, update and delete) operations for data entries stored in your regions.

- **Working with Queries**

  VMware GemFire supports the use of queries to extract data from its regions. Using REST APIs, you can create and execute either prepared or ad-hoc queries on VMware GemFire regions. You can also update and delete prepared queries.

- **Working with Functions**

  VMware GemFire REST APIs support the discovery and execution of predefined VMware GemFire functions on your cluster deployments.

Working with Regions

The VMware GemFire REST APIs provide basic CRUD (create, read, update and delete) operations for data entries stored in your regions.

Regions are the resources of the VMware GemFire REST API. Each region represents a resource or a collection of resources.

You cannot create or delete the regions themselves with the REST APIs, but you can work with the data stored within predefined VMware GemFire regions. Use the gfsh command utility to add, configure or delete regions in your VMware GemFire deployment. Any additions or modifications to regions made through `gfsh` are then accessible by the REST APIs.

Listing Available Regions

The main resource endpoint to the VMware GemFire API is GET /geode/v1. Use this endpoint to discover which regions are available in your cluster.

Example call:

```
curl -i http://localhost:7070/geode/v1
```

Example response:

```
Accept: application/json
Response Payload: application/json

200 OK
Server: Apache-Coyote/1.1
Location: http://localhost:7070/geode/v1
Content-Type: application/json
Transfer-Encoding: chunked
Date: Sat, 18 Jan 2014 20:05:47 GMT
```

```
{
    "regions": [
        {
            "name": "customers",
            "type": "REPLICATE",
            "key-constraint": "java.lang.String",
            "value-constraint": "org.apache.geode.pdx.PdxInstance"
        },
        {
            "name": "items",
            "type": "REPLICATE",
            "key-constraint": null,
            "value-constraint": null
        },
        {
            "name": "orders",
            "type": "PARTITION",
            "key-constraint": null,
            "value-constraint": null
        },
        {
            "name": "primitiveKVStore",
            "type": "PARTITION",
            "key-constraint": null,
            "value-constraint": null
        },
        {
            "name": "empty_region",
            "type": "EMPTY",
            "key-constraint": "java.lang.String",
            "value-constraint": "org.apache.geode.pdx.PdxInstance"
        }
    ]
}
```

Each region listed in the response includes the following region attributes:

- **name**. Name of the region.
- **type**. Type of region. For example, REPLICATE, PARTITION, or EMPTY. See Region Types for more information.
- **key-constraint**. If defined, the fully qualified class name of the key's type. Otherwise, null.
- **value-constraint**. If defined, the fully qualified class name of the value's type. Otherwise, null.

If no resources (regions) are available in the cluster, the call returns a 404 NOT FOUND error.

Reading Region Data

You can read data from a region by using any of the following REST-enabled mechanisms:

- **GET /geode/v1/{region}?limit=ALL** - Read all entries in a region
- **GET /geode/v1/{region}?limit=*N*** - Read a limited number of entries in a region
- **GET /geode/v1/{region}/keys** - List all keys in a region
- **GET /geode/v1/{region}/{keys}** - Read data for specific key or keys in a region

Reading Entries

To read entries in a region, use the following REST endpoint:

```
GET /geode/v1/{region}?[limit={<number>|ALL}]
```

For example:

```
http://localhost:7070/geode/v1/items
```

To read all entries in the region, specify instead:

```
http://localhost:7070/geode/v1/items?limit=ALL
```

To read 80 entries from the region, specify:

```
http://localhost:7070/geode/v1/items?limit=80
```

Setting the limit parameter is optional. If you do not specify a limit, the request will return 50 results by default. The returned order of the results is not guaranteed; however, the response values (JSON) are specified in the same order as the list of comma-separated keys returned in the “Content-location” header.

Reading Keys

To list all keys in a region, use the following endpoint:

```
/geode/v1/{region}/keys
```

For example:

```
http://localhost:7070/geode/v1/items/keys
```

You can use the returned key to perform additional operations on the keys such as read, update or delete their values.

Reading Entries By Key

To obtain data for a specific key or set of keys, use the following endpoints:

```
GET /geode/v1/{region}/{key1}
```

or

```
GET /geode/v1/{region}/{key1},{key2},...,{keyN}
```

where you specify keys in a comma-delimited list.

For example:

```
http://localhost:7070/geode/v1/items/1

http://localhost:7070/geode/v1/items/1,3,5
```

If one or more of the keys you provide in the list of keys is missing from the region, you will receive a 400 BAD STATUS error response.

If you are providing multiple keys, you can also use the `ignoreMissingKey=true` parameter to prevent 400 errors. Any non-existing keys will instead return a null response. For example:

```
http://localhost:7070/geode/v1/items/1,3,5?ignoreMissingKey=true
```

Adding or Modifying Region Data

To add data to a region, you have several options:

- Create new a brand new entry (both key and value) in a region
- Update or insert a value for a key (if the key does not exist, it will be created)
- Update (replace) data for a key if and only if the key already exists in region
- Compare existing value for a key before replacing value
- Update or insert multiple values in the region for a set of provided keys

Adding entries

To add a new key and value to a region, you can use the following endpoint:

```
POST /geode/v1/{region}?key=<key>
```

This endpoint only puts the entry into the region **if the specified key does not already exist**. If the key already exists, the request will fail with a 409 CONFLICT error.

Note: If you do not specify a key for this request, a String representation of a numerical key will be generated automatically for you.

Specify the value for the new entry in the request body. For example:

```
http://localhost:7070/geode/v1/orders?key=2
```

```
Request Payload: application/json
POST /geode/v1/orders?key=2

Accept: application/json
Content-Type: application/json
{
    "@type":  "org.apache.geode.web.rest.domain.Order",
     "purchaseOrderNo": 112,
     "customerId": 1012,
     "description":  "Purchase Order for myCompany",
     "orderDate":  "02/10/2014",
     "deliveryDate":  "02/20/2014",
     "contact":  "John Doe",
     "email":  "John.Doe@example.com",
     "phone":  "01-2048096",
     "totalPrice": 225,
     "items": [
        {
             "itemNo": 1,
             "description":  "Product2, PartA",
             "quantity": 10,
             "unitPrice": 5,
             "totalPrice": 50
        },
        {
             "itemNo": 2,
             "description":  "Product2, PartB",
             "quantity": 20,
             "unitPrice": 20,
             "totalPrice": 400
        }
    ]
}
```

Note that in the example above, `@type` is used to declare the value-constraint for the new entry. This declaration is required to provide interoperability between Java cache clients and REST clients.

Alternately, you can also use the following endpoint to create a new entry:

```
PUT /gemfire/v1/{region}/{key}
```

This endpoint will add the entry if the key does not exist. If the key already exists, the operation will update the entry.

```
http://localhost:7070/geode/v1/orders/2
```

```
Request Payload: application/json
PUT /geode/v1/orders/2
Request Payload: application/json
Content-Type: application/json
Accept: application/json
{
    "@type": "org.apache.geode.web.rest.domain.Order",
     "purchaseOrderNo": 1121,
     "customerId": 1012,
     "description":  "Order for XYZ Corp",
     "orderDate":  "02/10/2014",
     "deliveryDate":  "02/20/2014",
     "contact":  "Pie Doe",
     "email":  "pie.doe@example.com",
     "phone":  "01-2048096",
     "totalPrice": 225,
     "items": [
        {
             "itemNo": 1,
             "description": "Product-100",
             "quantity": 10,
             "unitPrice": 5,
             "totalPrice": 50
        }
    ]
}
```

Modifying existing entries

VMware GemFire provides three different options for this type of operation. To update a value for the key, you can use:

```
PUT /gemfire/v1/{region}/{key}
PUT /gemfire/v1{region}/{key}?op=REPLACE
PUT /gemfire/v1{region}/{key}?op=CAS
```

If you do not specify a parameter to the PUT operation, the request which will add or update the entry regardless of whether the key already exists in the region. See the example above.

```
http://localhost:7070/geode/v1/orders/2
```

If you specify the op=REPLACE parameter, the request which will explicitly perform a Cache replace operation and will verify that the key exists before replacing the value. If the key does not exist in the specified region, you will receive a 404 NOT FOUND error. This operation is idempotent, meaning multiple identical requests will have the same effect as the initial request.

```
http://localhost:7070/geode/v1/orders/2?op=REPLACE
```

If you specify the op=CAS parameter, the value will only be replaced with the `@new` value only if the specified `@old` value matches the current value of the key in the region. If the `@old` value does not match the current value, then a 409 CONFLICT error is thrown. If you receive a 409 CONFLICT error, you can call the `GET /geode/v1/{region}/{key}` endpoint to get an updated copy of the value. This operation is not idempotent and multiple identical requests will not have the same effect as the initial request. You can use this type of REST call to achieve a similar effect as optimistic locking.

```
http://localhost:7070/geode/v1/orders/2?op=REPLACE
```

```
Request Payload: application/json

PUT /geode/v1/orders/2?op=CAS

Accept: application/json
Content-Type: application/json
{
    "@old": {
         "@type":  "org.apache.geode.web.rest.domain.Order",
         "purchaseOrderNo": 1121,
         "customerId": 1012,
         "description":  "Order for  XYZ Corp",
         "orderDate":  "02/10/2014",
         "deliveryDate":  "02/20/2014",
         "contact":  "Jelly Bean",
         "email":  "jelly.bean@example.com",
         "phone":  "01-2048096",
         "items": [
            {
                 "itemNo": 1,
                 "description":  "Product-100",
                 "quantity": 12,
                 "unitPrice": 5,
                 "totalPrice": 60
            }
```

```
        ],
         "totalPrice": 225
    },
     "@new ": {
         "@type":  "org.apache.geode.web.rest.domain.Order",
         "purchaseOrderNo": 1121,
         "customerId": 1013,
         "description":  "Order for  New Corp",
         "orderDate":  "02/10/2014",
         "deliveryDate":  "02/25/2014",
         "contact":  "Vanilla Bean",
         "email":  "vanillabean@example.com",
         "phone":  "01-2048096",
         "items": [
            {
                 "itemNo": 12345,
                 "description":  "part 123",
                 "quantity": 12,
                 "unitPrice": 29.99,
                 "totalPrice": 149.95
            }
        ],
         "totalPrice": 149.95
    }
}
```

Adding or updating multiple values for a set of keys

To update multiple values for keys, use:

```
PUT /geode/v1/{region}/{key1},{key2},...,{keyN}
```

This REST call will update any keys that already exist and insert values for any keys that do not exist in the region.

Deleting Region Data

There are three options for deleting data in a region using REST APIs:

- Delete all the data in the region. Limited to replicated regions only; not available for partitioned regions.
- Delete the data associated with a particular key
- Delete the data associated with a set of keys

Deleting all data

To delete all data in the region, use the following endpoint:

```
DELETE /geode/v1/{region}
```

For example:

```
http://localhost:7070/geode/v1/items
```

Note that this does not delete the region itself, but instead all the entries in the region.

Deleting data based on key

Use:

```
DELETE /geode/v1/{region}/{key}
```

or

```
DELETE /geode/v1/{region}/{key}{key1},{key2},...{keyN}
```

If any of the supplied keys are not found in the region, the request will fail and return a 404 NOT FOUND ERROR.

Working with Queries

VMware GemFire supports the use of queries to extract data from its regions. Using REST APIs, you can create and execute either prepared or ad-hoc queries on VMware GemFire regions. You can also update and delete prepared queries.

Listing Queries

To find out which predefined and named queries are available in your deployment, use the following endpoint:

```
GET /geode/v1/queries
```

All queries that have been predefined and assigned IDs in VMware GemFire are listed.

Creating a New Query

To create a query, use the following endpoint:

```
POST /geode/v1/queries?id=<queryId>&q=<OQL-statement>
```

Here are some examples:

```
http://localhost:7070/geode/v1/queries?id=selectOffers&q="SELECT DISTINCT c FROM /cust
omers c, /orders o WHERE o.totalprice < $1 AND c.customerId = o.customerId"
```

Note: The query must be provided as a URL parameter. You cannot specify OQL in the request body at this time.

You can specify query bind parameters ($1) in your predefined queries and then pass in values at runtime.

To update this query at a later time, use the PUT operation described below.

Executing a Prepared Query

To run a prepared query, use:

```
POST /geode/v1/queries/{queryId}
```

Specify the queryId in the URL. All query argument must be passed in the request body. For example:

```
http://localhost:7070/geode/v1/queries/selectOrders
```

```
Request Payload: OQL bind parameter values HTTP message body of media type applicatio
n/json
POST /geode/v1/queries/selectOrders
Accept: application/json
Content-Type: application/json

[
    {
         "@type": "int",
         "@value": 2
    },
    {
         "@type": "double",
         "@value": 110.00
    }
]

Response Payload: application/json

200 OK
Content-Length: <#-of-bytes>
Content-Type: application/json

[
    {
        "description": "Purchase order for company - B",
        "totalPrice": 350,
        "purchaseOrderNo": 1112,
        "customerId": 102,
        "deliveryDate": "Thu Feb 20 00:00:00 IST 2014",
        "contact": "John Doe",
        "email": "jDoe@example.com",
        "phone": "01-2048096",
        "items": [
            {
                "description": "Product-AAAA",
                "quantity": 10,
                "itemNo": 1,
                "unitPrice": 20,
                "totalPrice": 200,
                "type-class": "org.apache.geode.web.rest.domain.Item"
            },
            {
                "description": "Product-BBB",
                "quantity": 15,
                "itemNo": 2,
                "unitPrice": 10,
                "totalPrice": 150,
                "type-class": "org.apache.geode.web.rest.domain.Item"
            }
        ],
        "orderDate": "Mon Feb 10 00:00:00 IST 2014",
        "type-class": "org.apache.geode.web.rest.domain.Order"
    },
    {...},
    {...}
}
```

Another example:

```
http://localhost:7070/geode/v1/queries/selectOrders
```

```
Request Payload: OQL bind parameter values HTTP message body of media type applicatio
n/json
POST /geode/v1/queries/selectCustomer
Accept: application/json
Content-Type: application/json
{
    "args": [
        {
            "@type": "int",
            "@value": 101
        }
    ]
}

Response-Payload: application/json

200 Ok
Content-Length: 140
Content-Type: application/json
[
    {
        "firstName": "Jane",
        "lastName": "Doe",
        "customerId": 101,
        "type-class": "org.apache.geode.web.rest.domain.Customer"
    }
]
```

Modifying a Prepared Query

To modify an existing prepared query, use the following endpoint:

```
PUT /geode/v1/queries/{queryId}
```

Here are some examples:

```
http://localhost:7070/geode/v1/queries/selectOffers&q="SELECT DISTINCT c FROM /custome
rs c, /orders o WHERE o.totalprice < $1 AND c.customerId = o.customerId"
```

You can specify query bind parameters ($1) in your predefined queries, and then pass in values at runtime.

A PUT operation will only succeed if the specified queryId already exists (for example, created with the POST operation above.) If the queryId does already exist, you will receive a 404 response - “Named query (selectKey456) does not exist!”

Deleting a Prepared Query

To delete an existing prepared query, use the following endpoint:

```
DELETE /geode/v1/queries/{queryId}
```

Executing an Ad-Hoc Query

To run an unnamed query, use the following endpoint:

```
GET /geode/v1/queries/adhoc?q=<OQL-statement>
```

Provide the OQL query string directly in the URL enclosed in single quotes.

For example:

```
http://localhost:7070/geode/v1/queries/adhoc?q="SELECT * FROM /customers"
```

Working with Functions

VMware GemFire REST APIs support the discovery and execution of predefined VMware GemFire functions on your cluster deployments.

Before you can access functions using REST APIs, you must have already defined and registered the functions in your VMware GemFire deployment. Additionally, any domain objects that are being accessed by the functions must be available on the CLASSPATH of the server running the REST endpoint service.

You can do the following with functions:

- List all functions available in the VMware GemFire cluster.
- Execute a function, optionally specifying the region and members and/or member groups that are targeted by the function

Listing Functions

To list all functions that are currently registered and deployed in the VMware GemFire cluster, use the following endpoint:

```
GET /geode/v1/functions
```

The list of returned functions includes the functionId, which you can use to execute the function as described in the next section.

Executing Functions

To execute a function on a VMware GemFire cluster, use the following endpoint:

```
POST /geode/v1/functions/{functionId}?[&onRegion=regionname|&onMembers=member1,member
2,...,memberN|&onGroups=group1,group2,...,groupN]
```

You have the option to target a specific region and members or member groups when executing your function. If you do not specify these parameters, the function will execute on all members that are hosting data in the entire cluster by default. Function arguments are passed in the request body.

For example:

```
http://localhost:7070/geode/v1/functions/AddFreeItemToOrders
```

```
Request Payload: application/json
POST /geode/v1/functions/AddFreeItemToOrders
Accept: application/json
Content-Type: application/json

[
    {
        "@type": "double",
        "@value": 210
    },
    {
        "@type": "org.apache.geode.web.rest.domain.Item",
        "itemNo": "599",
        "description": "Part X Free on Bumper Offer",
        "quantity": "2",
        "unitprice": "5",
        "totalprice": "10.00"
     }
]
```

In the above example, the `Item` domain object must be in the CLASSPATH of all members receiving the function. If the object is not defined, the function will fail with an `Internal Server` error. Look for `ClassNotFoundException`s in the stack trace.

Sample REST Applications

This section provides examples that illustrate how multiple clients, both REST and native, can access the same VMware GemFire region data.

Note: You must set PDX read-serialized to true when starting the cache server to achieve interoperability between different clients. See Setup and Configuration for instructions on starting up REST-enabled cache servers.

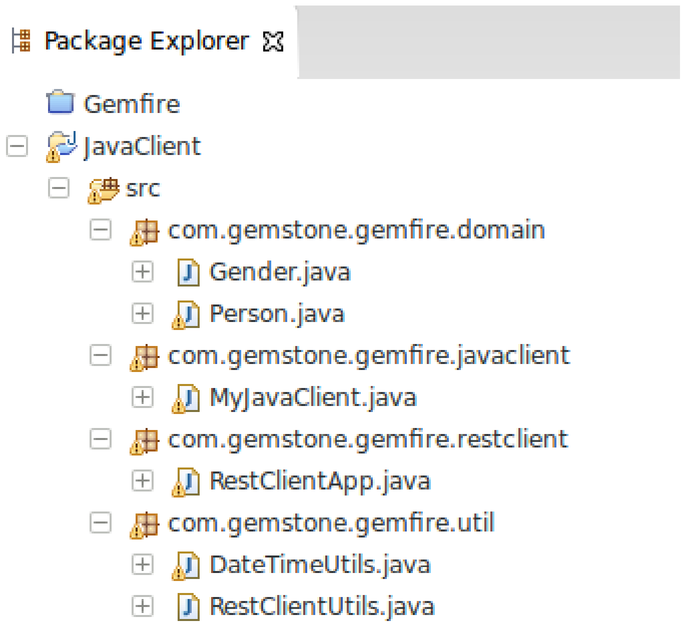
The following examples demonstrate the following:

1. A Java REST client creates a Person object on key 1. This client references the following supporting examples (also provided):
    1. VMware GemFire cache client
    2. REST client utility

3. Date Time utility
4. Person class
5. Gender class

2. A Ruby REST client also gets data for key 1 and updates it.
3. A Python REST Client demonstrates the creation and modification of objects. **Note:** An additional Python REST client reference application is available here: https://github.com/gemfire/py-gemfire-rest.

The following Java examples assume a project directory structure similar to the following:



```
Gemfire
JavaClient
  src
    com.gemstone.gemfire.domain
      Gender.java
      Person.java
    com.gemstone.gemfire.javaclient
      MyJavaClient.java
    com.gemstone.gemfire.restclient
      RestClientApp.java
    com.gemstone.gemfire.util
      DateTimeUtils.java
      RestClientUtils.java
```

#1. REST Java Client (RestClientApp.java)

```
package org.apache.geode.restclient;

 import org.springframework.http.HttpHeaders;
 import org.springframework.http.MediaType;
 import org.springframework.http.HttpMethod;
 import org.springframework.http.HttpEntity;
 import org.springframework.http.ResponseEntity;
 import org.springframework.web.client.HttpClientErrorException;
 import org.springframework.web.client.HttpServerErrorException;

 import org.apache.geode.util.RestClientUtils;

 import java.util.ArrayList;
 import java.util.List;

@SuppressWarnings( "unused")
 public class RestClientApp  {
   private static final String PEOPLE_REGION =  "/People";

   private static final String PERSON1_AS_JSON =  "{"
          +  "\"@type\ ": \"org.apache.geode.domain.Person\ "," +  "\"id\ ": 1,"
       +  " \"firstName\ ": \"Jane\ "," +  " \"middleName\ ": \"H\ ","
       +  " \"lastName\ ": \"Doe1\ "," +  " \"birthDate\ ": \"04/12/1983\ ","
```

```
        +  "\"gender\ ": \"MALE\ "" + "}";

    public static void main( final String... args)  throws Exception {
     doCreate(PEOPLE_REGION,  "1");
      System.out.println( "Programme has run successfully...!");
   }

    private static  HttpHeaders setAcceptAndContentTypeHeaders(){
     List<MediaType> acceptableMediaTypes =  new ArrayList<MediaType>();
     acceptableMediaTypes.add(MediaType.APPLICATION_JSON);

     HttpHeaders headers =  new HttpHeaders();
     headers.setAccept(acceptableMediaTypes);
     headers.setContentType(MediaType.APPLICATION_JSON);
      return headers;
   }

    private static void doCreate( final String regionNamePath,  final String key) {
     HttpHeaders headers =  setAcceptAndContentTypeHeaders();
     HttpEntity< String> entity =  new HttpEntity< String>(PERSON1_AS_JSON, headers);
      try {
       ResponseEntity< String> result = RestClientUtils.getRestTemplate().exchange(
              "http://localhost:8080/geode/v1/People?key=1" , HttpMethod.POST,
             entity,  String.class);

        System.out.println( "STATUS_CODE = " + result.getStatusCode().value());
        System.out.println( "HAS_BODY = " + result.hasBody());
        System.out.println( "LOCATION_HEADER = " + result.getHeaders().getLocation().to
String());
     }  catch (HttpClientErrorException e) {
        System.out.println( "Http Client encountered error, msg:: " + e.getMessage());
     }  catch(HttpServerErrorException se) {
        System.out.println( "Server encountered error, msg::" + se.getMessage());
     }  catch (Exception e) {
        System.out.println( "Unexpected ERROR...!!");
     }
   }
}
```

#1a. VMware GemFire Cache Java Client (MyJavaClient.java)

```
package org.apache.geode.javaclient;

 import java.util.Calendar;
 import java.util.HashMap;
 import java.util.Map;

 import org.apache.geode.cache.Region;
 import org.apache.geode.cache.client.ClientCache;
 import org.apache.geode.cache.client.ClientCacheFactory;
 import org.apache.geode.cache.client.ClientRegionFactory;
 import org.apache.geode.cache.client.ClientRegionShortcut;
 import org.apache.geode.domain.Gender;
 import org.apache.geode.domain.Person;
 import org.apache.geode.pdx.PdxInstance;
 import org.apache.geode.util.DateTimeUtils;
 public class MyJavaClient {

   public static void main( String[] args) {
    ClientCacheFactory cf =  new ClientCacheFactory().addPoolServer( "localhost", 4040
5);
    ClientCache cache = cf.setPdxReadSerialized( true).create();
    ClientRegionFactory rf = cache.createClientRegionFactory(ClientRegionShortcut.PROX
Y);

    Region region = rf.create( "People");

     //Get data on key "1" , update it and put it again in cache
    Person actualObj =  null;
     Object obj = region.get( "1");
     if(obj  instanceof PdxInstance){
       System.out.println( "Obj is PdxInstance");
      PdxInstance pi = (PdxInstance)obj;
       Object obj2 = pi.getObject();
       if(obj2  instanceof Person){
        actualObj = (Person)obj2;
         System.out.println( "Received Person :" + actualObj.toString());
      } else {
         System.out.println( "Error: obj2 is expected to be of type Person");
      }
    } else {
       System.out.println( "Error: obj is expected to be of type PdxInstance");
    }

     //update the received object and put it in cache
 if(actualObj !=  null){
      actualObj.setFirstName( "Jane_updated");
      actualObj.setLastName( "Doe_updated");
      region.put( "1", actualObj);
    }

     //Add/putAll set of person objects
 final Person person2 =  new Person(102L,  "Sachin",  "Ramesh",  "Tendulkar", DateTime
Utils.createDate(1975, Calendar.DECEMBER, 14), Gender.MALE);
     final Person person3 =  new Person(103L,  "Saurabh",  "Baburav",  "Ganguly", Date
TimeUtils.createDate(1972, Calendar.AUGUST, 29), Gender.MALE);
     final Person person4 =  new Person(104L,  "Rahul",  "subrymanyam",  "Dravid", Dat
eTimeUtils.createDate(1979, Calendar.MARCH, 17), Gender.MALE);
     final Person person5 =  new Person(105L,  "Jhulan",  "Chidambaram",  "Goswami", D
ateTimeUtils.createDate(1983, Calendar.NOVEMBER, 25), Gender.FEMALE);
     final Person person6 =  new Person(101L,  "Rahul",  "Rajiv",  "Gndhi", DateTimeUt
ils.createDate(1970, Calendar.MAY, 14), Gender.MALE);
     final Person person7 =  new Person(102L,  "Narendra",  "Damodar",  "Modi", DateTi
meUtils.createDate(1945, Calendar.DECEMBER, 24), Gender.MALE);
     final Person person8 =  new Person(103L,  "Atal",  "Bihari",  "Vajpayee", DateTim
eUtils.createDate(1920, Calendar.AUGUST, 9), Gender.MALE);
     final Person person9 =  new Person(104L,  "Soniya",  "Rajiv",  "Gandhi", DateTime
Utils.createDate(1929, Calendar.MARCH, 27), Gender.FEMALE);
     final Person person10 =  new Person(104L,  "Priyanka",  "Robert",  "Gandhi", Date
TimeUtils.createDate(1973, Calendar.APRIL, 15), Gender.FEMALE);
     final Person person11 =  new Person(104L,  "Murali",  "Manohar",  "Joshi", DateTi
meUtils.createDate(1923, Calendar.APRIL, 25), Gender.MALE);
     final Person person12 =  new Person(104L,  "Lalkrishna",  "Parmhansh",  "Advani",
DateTimeUtils.createDate(1910, Calendar.JANUARY, 01), Gender.MALE);
```

```
     final Person person13 =  new Person(104L,  "Shushma",  "kumari",  "Swaraj", DateT
imeUtils.createDate(1943, Calendar.AUGUST, 10), Gender.FEMALE);
     final Person person14 =  new Person(104L,  "Arun",  "raman",  "jetly", DateTimeUt
ils.createDate(1942, Calendar.OCTOBER, 27), Gender.MALE);
     final Person person15 =  new Person(104L,  "Amit",  "kumar",  "shah", DateTimeUti
ls.createDate(1958, Calendar.DECEMBER, 21), Gender.MALE);
     final Person person16 =  new Person(104L,  "Shila",  "kumari",  "Dixit", DateTime
Utils.createDate(1927, Calendar.FEBRUARY, 15), Gender.FEMALE);

    Map< String,  Object> userMap =  new HashMap< String,  Object>();
    userMap.put( "2", person6);
    userMap.put( "3", person6);
    userMap.put( "4", person6);
    userMap.put( "5", person6);
    userMap.put( "6", person6);
    userMap.put( "7", person7);
    userMap.put( "8", person8);
    userMap.put( "9", person9);
    userMap.put( "10", person10);
    userMap.put( "11", person11);
    userMap.put( "12", person12);
    userMap.put( "13", person13);
    userMap.put( "14", person14);
    userMap.put( "15", person15);
    userMap.put( "16", person16);

     //putAll all person
    region.putAll(userMap);

     System.out.println( "successfully Put set of Person objects into the cache");
  }

}
```

#1b. REST Client Utilities (RestClientUtils.java)

```
package org.apache.geode.util;


 import java.net.URI;
 import java.text.SimpleDateFormat;
 import java.util.ArrayList;
 import java.util.List;
 import org.springframework.http.converter.ByteArrayHttpMessageConverter;
 import org.springframework.http.converter.HttpMessageConverter;
 import org.springframework.http.converter.StringHttpMessageConverter;
 import org.springframework.http.converter.json.Jackson2ObjectMapperFactoryBean;
 import org.springframework.http.converter.json.MappingJackson2HttpMessageConverter;
 import org.springframework.web.client.RestTemplate;
 import org.springframework.web.util.UriComponentsBuilder;

 public class RestClientUtils {

   public static final String BASE_URL =  "http://192.0.2.0:8080" ;
   public static final String GEODE_REST_API_CONTEXT =  "/geode";
   public static final String GEODE_REST_API_VERSION =  "/v1";
   public static final URI GEODE_REST_API_WEB_SERVICE_URL = URI
      .create(BASE_URL + GEODE_REST_API_CONTEXT + GEODE_REST_API_VERSION);
   public static RestTemplate restTemplate;
   public static RestTemplate getRestTemplate() {
     if (restTemplate ==  null) {
      restTemplate =  new RestTemplate();
       final List<HttpMessageConverter<?>> messageConverters =  new ArrayList<HttpMess
ageConverter<?>>();

      messageConverters.add( new ByteArrayHttpMessageConverter());
      messageConverters.add( new StringHttpMessageConverter());
      messageConverters.add(createMappingJackson2HttpMessageConverter());

      restTemplate.setMessageConverters(messageConverters);
    }
     return restTemplate;
  }

   public static HttpMessageConverter< Object> createMappingJackson2HttpMessageConvert
er() {
     final Jackson2ObjectMapperFactoryBean objectMapperFactoryBean =  new Jackson2Obje
ctMapperFactoryBean();
    objectMapperFactoryBean.setFailOnEmptyBeans( true);
    objectMapperFactoryBean.setIndentOutput( true);
    objectMapperFactoryBean.setDateFormat( new SimpleDateFormat( "MM/dd/yyyy"));
    objectMapperFactoryBean
        .setFeaturesToDisable(com.fasterxml.jackson.databind.DeserializationFeature.FA
IL_ON_UNKNOWN_PROPERTIES);
    objectMapperFactoryBean
        .setFeaturesToEnable(
            com.fasterxml.jackson.core.JsonParser.Feature.ALLOW_COMMENTS,
            com.fasterxml.jackson.core.JsonParser.Feature.ALLOW_SINGLE_QUOTES,
            com.fasterxml.jackson.databind.DeserializationFeature.ACCEPT_EMPTY_STRING_
AS_NULL_OBJECT);
    objectMapperFactoryBean.afterPropertiesSet();

     final MappingJackson2HttpMessageConverter httpMessageConverter =  new MappingJack
son2HttpMessageConverter();
    httpMessageConverter.setObjectMapper(objectMapperFactoryBean.getObject());
     return httpMessageConverter;
  }

   public static URI toUri( final String... pathSegments) {
     return toUri(GEODE_REST_API_WEB_SERVICE_URL, pathSegments);
  }

   public static URI toUri( final URI baseUrl,  final String... pathSegments) {
     return UriComponentsBuilder.fromUri(baseUrl).pathSegment(pathSegments)
        .build().toUri();
  }
}
```

#1c. Date and Time Utilities (DateTimeUtils.java)

```
package org.apache.geode.util;


 import java.text.SimpleDateFormat;
 import java.util.Calendar;
 import java.util.Date;

/**
 * The DateTimeUtils class is a utility class  for working with dates and times.
 */
@SuppressWarnings( "unused")
 public abstract class DateTimeUtils {

   public static Calendar createCalendar( final int year,  final int month,  final int
day) {
     final Calendar dateTime = Calendar.getInstance();
    dateTime.clear();
    dateTime.set(Calendar.YEAR, year);
    dateTime.set(Calendar.MONTH, month);
    dateTime.set(Calendar.DAY_OF_MONTH, day);
     return dateTime;
  }

   public static Date createDate( final int year,  final int month,  final int day) {
     return createCalendar(year, month, day).getTime();
  }

   public static String format( final Date dateTime,  final String formatPattern) {
     return (dateTime !=  null ?  new SimpleDateFormat(formatPattern).format(dateTime)
:  null);
  }

}
```

#1d. Person Class (Person.java)

```
package org.apache.geode.domain;


 import java.util.Date;

 import org.apache.geode.internal.lang.ObjectUtils;
 import org.apache.geode.pdx.PdxReader;
 import org.apache.geode.pdx.PdxSerializable;
 import org.apache.geode.pdx.PdxWriter;

 import org.apache.geode.util.DateTimeUtils;

/**
 * The Person class is an abstraction modeling a person.
 */

 public class Person  implements PdxSerializable /*ResourceSupport  implements DomainO
bject< Long>*/  {

   private static final long serialVersionUID = 42108163264l;

   protected static final String DOB_FORMAT_PATTERN =  "MM/dd/yyyy";

   private Long id;

   private Date birthDate;
```

```
    private Gender gender;

    private String firstName;
    private String middleName;
    private String lastName;

    public Person() {
   }

    public Person( final Long id) {
      this.id = id;
   }

    public Person( final String firstName,  final String lastName) {
      this.firstName = firstName;
      this.lastName = lastName;
   }

    public Person( final Long id,  final String firstName,  final String middleName,  f
inal String lastName, Date date, Gender gender) {
      this.id = id;
      this.firstName = firstName;
      this.middleName = middleName;
      this.lastName = lastName;
      this.birthDate = date;
      this.gender = gender;
   }

    public Long getId() {
      return id;
   }

    public void setId( final Long id) {
      this.id = id;
   }

    public String getFirstName() {
      return firstName;
   }

    public void setFirstName( final String firstName) {
      this.firstName = firstName;
   }

    public String getLastName() {
      return lastName;
   }

    public void setLastName( final String lastName) {
      this.lastName = lastName;
   }

    public String getMiddleName() {
      return middleName;
   }

    public void setMiddleName( final String middleName) {
      this.middleName = middleName;
   }
```

```
    public Date getBirthDate() {
      return birthDate;
   }

    public void setBirthDate( final Date birthDate) {
      this.birthDate = birthDate;
   }

    public Gender getGender() {
      return gender;
   }

    public void setGender( final Gender gender) {
      this.gender = gender;
   }

   @Override
    public boolean equals( final Object obj) {
      if (obj ==  this) {
        return true;
     }

      if (!(obj  instanceof Person)) {
        return false;
     }

      final Person that = (Person) obj;

      return (ObjectUtils.equals( this.getId(), that.getId())
       || (ObjectUtils.equals( this.getBirthDate(), that.getBirthDate())
       && ObjectUtils.equals( this.getLastName(), that.getLastName())
       && ObjectUtils.equals( this.getFirstName(), that.getFirstName())));
   }

   @Override
    public int hashCode() {
      int hashValue = 17;
     hashValue = 37 * hashValue + ObjectUtils.hashCode(getId());
     hashValue = 37 * hashValue + ObjectUtils.hashCode(getBirthDate());
     hashValue = 37 * hashValue + ObjectUtils.hashCode(getLastName());
     hashValue = 37 * hashValue + ObjectUtils.hashCode(getFirstName());
      return hashValue;
   }

   @Override
    public String toString() {
      final StringBuilder buffer =  new StringBuilder( "{ type = ");
     buffer.append(getClass().getName());
     buffer.append( ", id = ").append(getId());
     buffer.append( ", firstName = ").append(getFirstName());
     buffer.append( ", middleName = ").append(getMiddleName());
     buffer.append( ", lastName = ").append(getLastName());
     buffer.append( ", birthDate = ").append(DateTimeUtils.format(getBirthDate(), DOB_F
ORMAT_PATTERN));
     buffer.append( ", gender = ").append(getGender());
     buffer.append( " }");
      return buffer.toString();
   }
```

```
  @Override
   public void fromData(PdxReader pr) {

    id = pr.readLong( "id");
    firstName = pr.readString( "firstName");
    middleName = pr.readString( "middleName");
    lastName = pr.readString( "lastName");
    birthDate = pr.readDate( "birthDate");
    gender = (Gender)pr.readObject( "gender");
  }

  @Override
   public void toData(PdxWriter pw) {
    pw.writeLong( "id", id);
    pw.writeString( "firstName", firstName);
    pw.writeString( "middleName", middleName);
    pw.writeString( "lastName", lastName);
    pw.writeDate( "birthDate", birthDate);
    pw.writeObject( "gender", gender);
  }

}
```

#1e. Gender Class (Gender.java)

```
package org.apache.geode.domain;

/**
 * The Gender  enum is a enumeration of genders (sexes).
 */

 public enum Gender {
  FEMALE,
  MALE
}
```

#2. Ruby REST Client (restClient.rb)

```
#!/usr/bin/ruby -w

puts "Hello, Ruby!";

# !/usr/bin/env ruby

require 'json'
require 'net/http'

class JsonSerializable

  def to_json
    hash = {}
    hash["@type"] = "org.apache.geode.web.rest.domain.Person"
    self.instance_variables.each do |var|
      if !var.to_s.end_with?("links")
        hash[var.to_s[1..-1]] = self.instance_variable_get var
      end
    end
    hash.to_json
  end

  def from_json! jsonString
    JSON.load(jsonString).each do |var, val|
      if !var.end_with?("type")
        self.instance_variable_set "@".concat(var), val
      end
    end
  end

end

class Person < JsonSerializable

  attr_accessor :id, :firstName, :middleName, :lastName, :birthDate, :gender

  def initialize(id = nil, firstName = nil, middleName = nil, lastName = nil )
    @id = id
    @firstName = firstName
    @middleName = middleName
    @lastName = lastName
    @birthDate = nil
    @gender = nil
  end

  def to_s
    s = "{ type = Person, id = #{@id}"
    s += ", firstName = #{@firstName}"
    s += ", middleName = #{@middleName}"
    s += ", lastName = #{@lastName}"
    s += ", birthDate = #{@birthDate}"
    s += ", gender = #{@gender}"
    s += "}"
  end

end

if __FILE__ == $0
  #p = Person.new(1, "Jon", "T", "Doe")
  #puts p
  #puts p.inspect
  #puts p.to_json

  uri = URI("http://localhost:8080/geode/v1/People/1");

  personJson = Net::HTTP::get(uri);

  # JSON from server
  puts "JSON read from Server for Person with ID 1...\n #{personJson}"

  p = Person.new
  p.from_json! personJson

  # print the Person to standard out
  puts "Person is...\n #{p}"
  p.id = 1
  p.firstName = "Jack"
  p.lastName = "Handy"
  p.gender = "MALE"

  # prints modified Person to standard out
  puts "Person modified is...\n #{p}"

  puts "JSON sent to Server for Person with ID 1...\n #{p.to_json}"

  Net::HTTP.start(uri.hostname, uri.port) do |http|
    putRequest = Net::HTTP::Put.new uri.path, { "Content-Type" => "application/json" }
    putRequest.body = p.to_json
    http.request(putRequest)
  end

end
```

Output from running the Ruby client:

```
prompt# ruby restClient.rb
Hello, Ruby!
JSON read from Server for Person with ID 1...
 {
  "@type" : "org.gopivotal.app.domain.Person",
  "id" : 1,
  "firstName" : "Jane_updated",
  "middleName" : "H",
  "lastName" : "Doe_updated",
  "gender" : "MALE",
  "birthDate" : "04/12/1983"
}
Person is...
 { type = Person, id = 1, firstName = Jane_updated, middleName = H, lastName = Doe_upd
ated, birthDate = 04/12/1983, gender = MALE}
Person modified is...
 { type = Person, id = 1, firstName = Jack, middleName = H, lastName = Handy, birthDat
e = 04/12/1983, gender = MALE}
JSON sent to Server for Person with ID 1...
 {"@type":"org.apache.geode.web.rest.domain.Person","id":1,"firstName":"Jack","middleN
ame":"H","lastName":"Handy","birthDate":"04/12/1983","gender":"MALE"}
```

#3. Python REST Client (restClient.py)

This example uses Python 3 and shows the creation and modification of objects. It uses one external library called `requests`, which is nearly ubiquitous and avoids having to use HTTP code.

```
#!/usr/bin/env python3

# This is simple, repetitive and assumes you have created a region called
# "demoRegion".

import sys
import json
import uuid
import requests

```

```
REGION = "demoRegion"
BASE_URI = "http://localhost:8080/geode/v1"

headers = {'content-type': 'application/json'}

person = {'type': 'Person',
          'firstName': 'John',
          'middleName': 'Q',
          'lastName': 'Public',
          'birthDate': '1 Jan 1900'}


def resource_uri(res=None, region=REGION):
    if res:
        return "%s/%s/%s" % (BASE_URI, region, res)
    return "%s/%s" % (BASE_URI, region)


print("[*] First, we'll empty out our demo region - DELETE %s" %
      requests.delete(resource_uri()))

r = requests.delete(resource_uri())
r.raise_for_status()

print("[*] Now, we'll create 5 demo objects")

keys = []

for i in range(1, 6):
    key = uuid.uuid1()

    keys.append(key)
    person['uuid'] = str(key)

    print("\t Creating object with key: POST %s" % key)
    r = requests.post(resource_uri(), data=json.dumps(person),
                      params={'key': key},
                      headers=headers)
    r.raise_for_status()

print("[*] List our keys - GET %s" % resource_uri("keys"))

r = requests.get(resource_uri("keys"))
print(r.text)

print("[*] Here's all our data - GET %s" % resource_uri())

r = requests.get(resource_uri())
print(r.text)

print("[*] Now each key one by one")

for key in keys:
    print("Fetching key - GET %s" % resource_uri(res=key))
    r = requests.get(resource_uri(res=key))
    print(r.text)

print("[*] Now grab one, change the first name to 'Jane' and save it")
print("  GET - %s" % resource_uri(res=keys[0]))
r = requests.get(resource_uri(res=keys[0]))
p = json.loads(r.text)
p['firstName'] = 'Jane'
print("  PUT - %s" % resource_uri(res=keys[0]))
r = requests.put(resource_uri(res=keys[0]), data=json.dumps(p),
                 headers=headers)

print("  GET - %s" % resource_uri(res=keys[0]))
r = requests.get(resource_uri(res=keys[0]))
print(r.text)
```

Troubleshooting and FAQ

This section provides troubleshooting guidance and frequently asked questions about VMware GemFire Developer REST APIs.

Checking if the REST API Service is Up and Running

Use the ping endpoint to verify whether the REST API server is available.

Use the `/geode/v1/ping` endpoint to check REST API server availability:

For example:

```
curl -i http://localhost:7070/geode/v1/ping
```

Example success response:

```
200 OK
```

If the server is not available, your client will receive an HTTP error code and message.

Key Types and JSON Support

When defining regions (your REST resources), you must only use scalar values for keys and also set value constraints in order to avoid producing JSON that cannot be parsed by VMware GemFire.

If VMware GemFire regions are not defined with scalar values as keys and value constraints, then you may receive the following error message (even though the JSON is technically valid) in your REST client applications:

```
Json doc specified in request body is malformed..!!'
```

For example, the following JSON documents are not supported by VMware GemFire:

Unsupported JSON Example 1

```
[
     1,
     [],
     [
         4,
         "hello",
         {}
     ],
     {
         "array": []
     }
]
```

Unsupported JSON Example 2

```
[[[[[[[[[[[[[[[[[[["Not too deep"]]]]]]]]]]]]]]]]]]]
```

Unsupported JSON Example 3

```
[1,2,3,"hello"]
```

Unsupported JSON Example 4

```
[
    "JSON Test Pattern pass1",
    {
        "object with 1 member": [
            "array with 1 element"
        ]
    },
    {},
    [],
    -42,
    true,
    false,
    null,
    {
        "integer": 1234567890,
        "real": -9876.54321,
        "e": 1.23456789e-13,
        "E": 1.23456789e+34,
        "": 2.3456789012e+76,
        "zero": 0,
        "one": 1,
        "space": " ",
        "quote": "\"",
        "backslash": "\",
        "controls": "\b\f\n\r\t",
        "slash": "/ & /",
        "alpha": "abcdefghijklmnopqrstuvwyz",
        "ALPHA": "ABCDEFGHIJKLMNOPQRSTUVWYZ",
        "digit": "0123456789",
        "0123456789": "digit",
        "special": "`1~!@#$%^&*()_+-={':[,]}|;.</>?",
        "true": true,
        "false": false,
        "null": null,
        "array": [],
        "object": {},
        "address": "50 St. James Street",
        "url": "http://www.JSON.org/",
        "comment": "// /* <!-- --",
        "# -- --> */": " ",
        " s p a c e d ": [
            1,
            2,
            3,
            4,
            5,
            6,
            7
        ],
        "compact": [
            1,
            2,
            3,
            4,
            5,
            6,
            7
        ],
        "jsontext": "{\"object with 1 member\":[\"array with 1 element\"]}",
        "quotes": "&#34; \" %22 0x22 034 &#x22;",
        "/\\"\b\f\n\r\t`1~!@#$%^&*()_+-=[]{}|;:',./<>?": "A key can be any string"
    },
    0.5,
    98.6,
    99.44,
    1066,
    10,
    1,
    0.1,
    1,
    2,
    2,
    "rosebud"
]
```

VMware GemFire REST API Reference

This section summarizes all available VMware GemFire REST API resources and endpoints.

Note: This documentation covers the **v1** release of VMware GemFire REST APIs for developing applications.

- **Region Endpoints**

  A VMware GemFire region is how VMware GemFire logically groups data within its cache. Regions stores data as entries, which are key-value pairs. Using the REST APIs you can read, add (or update), and delete region data.

- **Query Endpoints**

  VMware GemFire uses a query syntax based on OQL (Object Query Language) to query region data. Since VMware GemFire regions are key-value stores, values can range from simple byte arrays to complex nested objects.

- **Function Endpoints**

  VMware GemFire functions allows you to write and execute server-side transactions and data operations. These may include anything ranging from initializing components or third-party services or aggregating data.

- **Administrative Endpoints**

  Administrative endpoints provide management and monitoring functionality for the REST API interface.

Region Endpoints

A VMware GemFire region is how VMware GemFire logically groups data within its cache. Regions stores data as entries, which are key-value pairs. Using the REST APIs you can read, add (or update), and delete region data.

See also Data Regions for more information on working with regions.

- **GET /geode/v1**

  List all available resources (regions) in the VMware GemFire cluster.

- **GET /geode/v1/{region}**

  Read data for the region. The optional limit URL query parameter specifies the number of values from the Region that will be returned. The default limit is 50. If the user specifies a limit of “ALL”, then all entry values for the region will be returned.

- **GET /geode/v1/{region}/keys**

  List all keys for the specified region.

- **GET /geode/v1/{region}/{key}**

  Read data for a specific key in the region.

- **GET /geode/v1/{region}/{key1},{key2},…,{keyN}**

  Read data for multiple keys in the region.

- **HEAD /geode/v1/{region}**

  An HTTP HEAD request that returns region’s size (number of entries) within the HEADERS, which is a response without the content-body. Region size is specified in the pre-defined header named “Resource-Count”.

- **POST /geode/v1/{region}?key=<key>**

  Create (put-if-absent) data in region.

- **PUT /geode/v1/{region}/{key}**

  Update or insert (put) data for key in region.

- **PUT /geode/v1/{region}/{key1},{key2},…{keyN}**

  Update or insert (put) data for multiple keys in the region.

- **PUT /geode/v1/{region}/{key}?op=REPLACE**

Update (replace) data with key(s) if and only if the key(s) exists in region. The Key(s) must be present in the Region for the update to occur.

- **PUT /geode/v1/{region}/{key}?op=CAS**

  Update (compare-and-set) value having key with a new value if and only if the “@old” value sent matches the current value having key in region.

- **DELETE /geode/v1/{region}**

  Delete all entries in the region.

- **DELETE /geode/v1/{region}/{key}**

  Delete entry for specified key in the region.

- **DELETE /geode/v1/{region}/{key1},{key2},…{keyN}**

  Delete entries for multiple keys in the region.

GET /geode/v1

List all available resources (regions) in the VMware GemFire cluster.

Resource URL

```
http://<hostname_or_http-service-bind-address>:<http-service-port>/geode/v1
```

Parameters

None.

Example Request

```
GET /gemfire/v1/
Accept: application/json
```

Example Success Response

```
Response Payload: application/json

200 OK
Server: Apache-Coyote/1.1
Location: http://localhost:8080/geode/v1
Content-Type: application/json
Transfer-Encoding: chunked
Date: Sat, 18 Jan 2014 20:05:47 GMT
 {
    "regions": [
        {
            "name": "customers",
            "type": "REPLICATE",
            "key-constraint": "java.lang.String",
            "value-constraint": "org.apache.geode.pdx.PdxInstance"
        },
        {
            "name": "items",
            "type": "REPLICATE",
            "key-constraint": null,
            "value-constraint": null
        },
        {
            "name": "orders",
            "type": "PARTITION",
            "key-constraint": null,
            "value-constraint": null
        },
        {
            "name": "primitiveKVStore",
            "type": "PARTITION",
            "key-constraint": null,
            "value-constraint": null
        },
        {
            "name": "empty_region",
            "type": "EMPTY",
            "key-constraint": "java.lang.String",
            "value-constraint": "org.apache.geode.pdx.PdxInstance"
        }
    ]
}
```

Error Codes

Status Code	Description
404 NOT FOUND	No regions were found at the provided endpoint.
500 INTERNAL SERVER ERROR	Encountered error at server.

GET /geode/v1/{region}

Read data for the region. The optional limit URL query parameter specifies the number of values from the Region that will be returned. The default limit is 50. If the user specifies a limit of “ALL”, then all entry values for the region will be returned.

Resource URL

```
http://<hostname_or_http-service-bind-address>:<http-service-port>/geode/v1/{region}?
[limit={<number>|ALL}]
```

Parameters

Parameter	Description	Example Values
limit	**Optional.** Specify a limit to the number of region entries to return. If the limit parameter is not specified, the default is to return 50 results. Default: 50	ALL 80

Example Request

```
GET /geode/v1/orders/
Accept: application/json
```

Example Success Response

```
Response Payload: application/json

200 OK
Server: Apache-Coyote/1.1
Content-Location: http://localhost:8080/geode/v1/orders/3,1
Content-Type: application/json
Transfer-Encoding: chunked
Date: Sat, 18 Jan 2014 21:03:08 GMT

{
  "orders" : [ {
     "purchaseOrderNo" : 1112,
     "customerId" : 102,
     "description" :  "Purchase order for  company - B",
     "orderDate" :  "02/10/2014",
     "deliveryDate" :  "02/20/2014",
     "contact" :  "John Doe",
     "email" :  "John.Doe@example.com",
     "phone" :  "01-2048096",
     "items" : [ {
       "itemNo" : 1,
       "description" :  "Product-AAAA",
       "quantity" : 10,
       "unitPrice" : 20.0,
       "totalPrice" : 200.0
    }, {
       "itemNo" : 2,
       "description" :  "Product-BBB",
       "quantity" : 15,
       "unitPrice" : 10.0,
       "totalPrice" : 150.0
    } ],
     "totalPrice" : 350.0
  }, {
     "purchaseOrderNo" : 111,
     "customerId" : 101,
     "description" :  "Purchase order for  company - A",
     "orderDate" :  "01/10/2014",
     "deliveryDate" :  "01/20/2014",
     "contact" :  "Jane Doe",
     "email" :  "Jane.Doe@example.com",
     "phone" :  "020-2048096",
     "items" : [ {
       "itemNo" : 1,
       "description" :  "Product-1",
       "quantity" : 5,
       "unitPrice" : 10.0,
       "totalPrice" : 50.0
    }, {
       "itemNo" : 1,
       "description" :  "Product-2",
       "quantity" : 10,
       "unitPrice" : 15.5,
       "totalPrice" : 155.0
    } ],
     "totalPrice" : 205.0
  } ]
}
```

Error Codes

Status Code	Description
400 BAD REQUEST	Limit parameter **X** is not valid! The specified limit value must be ALL or an integer.
404 NOT FOUND	Returned if region does not exist.
500 INTERNAL ERROR	Error encountered at VMware GemFire server. Check the HTTP response body for a stack trace of the exception.

GET /geode/v1/{region}/keys

List all keys for the specified region.

Resource URL

```
http://<hostname_or_http-service-bind-address>:<http-service-port>/geode/v1/{region}/
{keys}
```

Parameters

None.

Example Request

```
GET /geode/v1/orders/keys
```

Example Success Response

```
Response Payload: application/json
```

```
200 OK
Server: Apache-Coyote/1.1
Location: http://localhost:8080/geode/v1/orders/keys
Content-Type: application/json
Transfer-Encoding: chunked
Date: Sat, 18 Jan 2014 21:20:05 GMT
{    "keys": [
        "1",
        "2",
        "3"
    ]
}
```

Error Codes

Status Codes	Description
404 NOT FOUND	Specified region does not exist.
405 METHOD NOT ALLOWED	Returned if any HTTP request method other than GET (for example, POST, PUT, DELETE, etc.) is used.
500 INTERNAL SERVER ERROR	Error encountered at VMware GemFire server. Check the HTTP response body for a stack trace of the exception.

GET /geode/v1/{region}/{key}

Read data for a specific key in the region.

Resource URL

```
http://<hostname_or_http-service-bind-address>:<http-service-port>/geode/v1/{region}/
{key}
```

Parameters

None.

Example Request

```
GET /geode/v1/orders/1

Request Payload: null
Accept: application/json
```

Example Responses

```
Response Payload: application/json

200 OK
Server: Apache-Coyote/1.1
```

```
Content-Location: http: //localhost:8080/geode/v1/orders/1
Content-Type: application/json
Transfer-Encoding: chunked
Date: Sat, 18 Jan 2014 21:27:59 GMT

{
   "purchaseOrderNo" : 111,
   "customerId" : 101,
   "description" :  "Purchase order for  company - A",
   "orderDate" :  "01/10/2014",
   "deliveryDate" :  "01/20/2014",
   "contact" :  "Jane Doe",
   "email" :  "Jane.Doe@example.com",
   "phone" :  "020-2048096",
   "items" : [ {
     "itemNo" : 1,
     "description" :  "Product-1",
     "quantity" : 5,
     "unitPrice" : 10.0,
     "totalPrice" : 50.0
  }, {
     "itemNo" : 1,
     "description" :  "Product-2",
     "quantity" : 10,
     "unitPrice" : 15.5,
     "totalPrice" : 155.0
  } ],
   "totalPrice" : 205.0
}
```

Error Codes

Status Code	Description
400 BAD REQUEST	Returned if the supplied key is not found in the region.
404 NOT FOUND	Returned if the region or specified key is not found.
500 INTERNAL SERVER ERROR	Error encountered at VMware GemFire server. Check the HTTP response body for a stack trace of the exception.

GET /geode/v1/{region}/{key1},{key2},…,{keyN}

Read data for multiple keys in the region.

Resource URL

```
http://<hostname_or_http-service-bind-address>:<http-service-port>/geode/v1/queries[?i
gnoreMissingKey=true]
```

Parameters

Parameter	Description	Example Values
ignoreMissingKey	**Optional**. Boolean to indicate whether to return non-existing keys as null responses. Use this to prevent 400 errors.	true false

Example Requests

```
GET /geode/v1/orders/1,2,13,4,5?ignoreMissingKey=true
Request Payload: null
Accept: application/json


GET /geode/v1/orders/1,3
Request Payload: null
Accept: application/json
```

Example Success Responses

```
Response Payload: application/json

200 OK
Server: Apache-Coyote/1.1
Content-Location: http://localhost:8080/geode/v1/orders/13,2,1,5,4
Content-Type: application/json
Transfer-Encoding: chunked
Date: Sat, 18 Jan 2014 21:39:04 GMT


{
    "orders": [
        null,
        {
            "@type": "org.apache.geode.rest.internal.web.controllers.Order",
            "purchaseOrderNo": 1112,
            "customerId": 102,
            "description": "Purchase order for company - B",
            "orderDate": "02/10/2014",
            "deliveryDate": "02/20/2014",
            "contact": "John Doe",
            "email": "john.doe@example.com",
            "phone": "01-2048096",
            "totalPrice": 350,
            "items": [
                {
                    "itemNo": 1,
                    "description": "Product-AAAA",
                    "quantity": 10,
                    "unitPrice": 20,
                    "totalPrice": 200
                },
                {
                    "itemNo": 2,
                    "description": "Product-BBB",
                    "quantity": 15,
                    "unitPrice": 10,
                    "totalPrice": 150
                }
            ]
        },
        {
            "@type": "org.apache.geode.rest.internal.web.controllers.Order",
            "purchaseOrderNo": 11101,
            "customerId": 101,
            "description": "Purchase order for company - A",
            "orderDate": "01/10/2014",
            "deliveryDate": "01/20/2014",
            "contact": "Jane Doe",
            "email": "jane.doe@example.com",
            "phone": "020-2048096",
            "totalPrice": 205,
            "items": [
                {
                    "itemNo": 1,
                    "description": "Product-1",
                    "quantity": 5,
                    "unitPrice": 10,
                    "totalPrice": 50
                },
                {
                    "itemNo": 3,
                    "description": "Product-3",
                    "quantity": 10,
                    "unitPrice": 100,
                    "totalPrice": 1000
                },
                {
                    "itemNo": 1,
                    "description": "Product-2",
                    "quantity": 10,
                    "unitPrice": 15.5,
                    "totalPrice": 155
                }
            ]
        },
        null,
        null
    ]
}


200 OK
Server: Apache-Coyote/1.1
Content-Location: http://localhost:8080/geode/v1/orders/3,1
Content-Type: application/json
Transfer-Encoding: chunked
Date: Sat, 18 Jan 2014 21:39:04 GMT

{
  "orders" : [ {
    "purchaseOrderNo" : 1112,
    "customerId" : 102,
    "description" : "Purchase order for company - B",
    "orderDate" : "02/10/2014",
    "deliveryDate" : "02/20/2014",
    "contact" : "John Doe",
    "email" : "john.doe@example.com",
    "phone" : "01-2048096",
    "items" : [ {
      "itemNo" : 1,
      "description" : "Product-AAAA",
      "quantity" : 10,
      "unitPrice" : 20.0,
      "totalPrice" : 200.0
    }, {
      "itemNo" : 2,
      "description" : "Product-BBB",
      "quantity" : 15,
      "unitPrice" : 10.0,
      "totalPrice" : 150.0
    } ],
    "totalPrice" : 350.0
  }, {
    "purchaseOrderNo" : 111,
    "customerId" : 101,
    "description" : "Purchase order for company - A",
    "orderDate" : "01/10/2014",
    "deliveryDate" : "01/20/2014",
    "contact" : "Jane Doe",
    "email" : "jane.doe@example.com",
    "phone" : "020-2048096",
    "items" : [ {
      "itemNo" : 1,
      "description" : "Product-1",
      "quantity" : 5,
      "unitPrice" : 10.0,
      "totalPrice" : 50.0
    }, {
      "itemNo" : 1,
      "description" : "Product-2",
      "quantity" : 10,
      "unitPrice" : 15.5,
      "totalPrice" : 155.0
    } ],
    "totalPrice" : 205.0
  } ]
}
```

Error Codes

Status Code	Description
400 BAD REQUEST	Returned if one or more of the supplied keys is not found in the region and ignoreMissingKey=false (default if parameter is not specified); returned if the specified value for the ignoreMissingKey parameter is a value other than 'true' or 'false'.
404 NOT FOUND	The specified region does not exist.

Example Error Response

```
GET /geode/v1/orders/1,2,13,4,5
Request Payload: null
Accept: application/json
Response Payload: application/json

400 BAD_REQUEST
Server: Apache-Coyote/1.1
Content-Type: application/json
Content-Length: 51
Date: Wed, 21 May 2014 11:24:00 GMT
Connection: close

"Requested keys [13,4,5] not exist in region [orders]"
```

Implementation Notes

The response body sets the region name as key to the array of values. For example “orders” : <VALUE_ARRAY>.

HEAD /geode/v1/{region}

An HTTP HEAD request that returns region’s size (number of entries) within the HEADERS, which is a response without the content-body. Region size is specified in the pre-defined header named “Resource-Count”.

Resource URL

```
http://<hostname_or_http-service-bind-address>:<http-service-port>/geode/v1/{region}
```

Parameters

None.

Example Request

```
Request Payload: null
HEAD /geode/v1/customers
Response Payload: null
```

Example Success Response

```
200 OK

Content-Length: <#-of-bytes>
Content-Type: application/json; charset=utf-8
...

Resource-Count: 8192
```

Error Codes

Status Code	Description
400 Bad Request	Returned if VMware GemFire throws an error while executing the request.
404 Resource Not Found	Region does not exist.
500 Internal Server Error	VMware GemFire has thown an error or exception.

POST /geode/v1/{region}?key=<key>

Create (put-if-absent) data in region.

Resource URL

```
http://<hostname_or_http-service-bind-address>:<http-service-port>/geode/v1/{region}?k
ey=<key>
```

Parameters

Parameter	Description	Example Values
key	**Optional.** If you do not specify a key in the URL, a key will automatically be generated for you.	123 myOrder

Example Request

```
Request Payload: application/json
POST /geode/v1/orders?key=2

Accept: application/json
Content-Type: application/json
{
    "@type":  "org.apache.geode.web.rest.domain.Order",
     "purchaseOrderNo": 112,
     "customerId": 1012,
     "description":  "Purchase Order for myCompany",
     "orderDate":  "02/10/2014",
     "deliveryDate":  "02/20/2014",
     "contact":  "John Doe",
     "email":  "John.Doe@example.com",
     "phone":  "01-2048096",
     "totalPrice": 225,
     "items": [
        {
             "itemNo": 1,
             "description":  "Product2, PartA",
             "quantity": 10,
             "unitPrice": 5,
             "totalPrice": 50
        },
        {
             "itemNo": 2,
             "description":  "Product2, PartB",
             "quantity": 20,
             "unitPrice": 20,
             "totalPrice": 400
        }
    ]
}
```

Example Success Response

```
Response Payload: null
201 CREATED
Location: http://localhost:8080/geode/v1/orders/2
```

Error Codes

Status Code	Description
400 BAD REQUEST	Returned if JSON content is malformed.
404 NOT FOUND	Returned if the specified region does not exist.
409 CONFLICT	Returned if the provided key already exists in the region.
500 INTERNAL SERVER ERROR	Error encountered at VMware GemFire server. Check the HTTP response body for a stack trace of the exception.

Example Error Response

```
409 Conflict
Location:http://localhost:8080/geode/v1/orders/2
Content-Type: application/json

{
     "purchaseOrderNo": 112,
     "customerId": 1012,
     "description":  "Purchase Order for myCompany",
     "orderDate":  "02/10/2014",
     "deliveryDate":  "02/20/2014",
     "contact":  "John Doe",
     "email":  "John.Doe@example.com",
     "phone":  "01-2048096",
     "totalPrice": 225,
     "items": [
        {
             "itemNo": 1,
             "description":  "Product2, PartA",
             "quantity": 10,
             "unitPrice": 5,
             "totalPrice": 50
        },
        {
             "itemNo": 2,
                "description":  "Product2, PartB",
                "quantity": 20,
                "unitPrice": 20,
                "totalPrice": 400
        }
    ]
}
```

PUT /geode/v1/{region}/{key}

Update or insert (put) data for key in region.

Resource URL

```
http://<hostname_or_http-service-bind-address>:<http-service-port>/geode/v1/{region}/
{key}
```

Parameters

See PUT /geode/v1/{region}/{key}?op=REPLACE and PUT /geode/v1/{region}/{key}?op=CAS.

Example Request

```
PUT /geode/v1/orders/2
Request Payload: application/json
Content-Type: application/json
Accept: application/json
{
    "@type": "org.apache.geode.web.rest.domain.Order",
     "purchaseOrderNo": 1121,
     "customerId": 1012,
     "description":  "Order for XYZ Corp",
     "orderDate":  "02/10/2014",
     "deliveryDate":  "02/20/2014",
     "contact":  "Pie Doe",
     "email":  "pie.doe@example.com",
     "phone":  "01-2048096",
     "totalPrice": 225,
     "items": [
        {
             "itemNo": 1,
             "description": "Product-100",
             "quantity": 10,
             "unitPrice": 5,
             "totalPrice": 50
        }
    ]
}
```

Example Success Response

```
Response Payload:  null

200 OK
```

Error Codes

Status Code	Description
400 BAD REQUEST	Returned if supplied key is an invalid format.
404 NOT FOUND	Returned if the region is not found.
500 INTERNAL SERVER ERROR	Error encountered at VMware GemFire server. Check the HTTP response body for a stack trace of the exception.

Implementation Notes

This operation is idempotent, meaning multiple identical requests should have the same effect as the initial request.

PUT /geode/v1/{region}/{key1},{key2},…{keyN}

Update or insert (put) data for multiple keys in the region.

Resource URL

```
http://<hostname_or_http-service-bind-address>:<http-service-port>/geode/v1/{region}/
{key},{key2},...{keyN}
```

Parameters

See PUT /geode/v1/{region}/{key}?op=REPLACE and PUT /geode/v1/{region}/{key}?op=CAS.

Example Request

```
Request Payload: application/json

PUT /geode/v1/orders/4,5

Accept: application/json
Content-Type: application/json
[
    {
        "@type":  "org.apache.geode.web.rest.domain.Order",
         "purchaseOrderNo": 555,
         "customerId": 5,
         "description":  "Order for 23 Corp",
         "orderDate":  "01/15/2014",
         "deliveryDate":  "02/20/2014",
         "contact":  "Jane Doe",
         "email":  "jane.doe@example.com",
         "phone":  "01-2048096",
         "items": [
            {
                 "itemNo": 321,
                 "description":  "part 21",
                 "quantity": 2,
                 "unitPrice": 49.95,
                 "totalPrice": 99.9
            }
        ],
         "totalPrice": 99.9
    },
    {
         "@type":  "org.apache.geode.web.rest.domain.Order",
         "purchaseOrderNo": 777,
         "customerId": 11,
         "description":  "Order for  MNP Corp",
         "orderDate":  "02/10/2014",
         "deliveryDate":  "02/20/2014",
         "contact":  "Trent Jones",
         "email":  "trent.jones@example.com",
         "phone":  "503-555-1221",
         "items": [
            {
                 "itemNo": 321,
                 "description":  "Product-21",
                 "quantity": 3,
                 "unitPrice": 49.95,
                 "totalPrice": 149.85
            }
        ],
         "totalPrice": 149.85
    }
]
```

Example Success Response

```
Response-payload: null

200 OK
```

Error Codes

Status Code	Description
400 BAD REQUEST	Returned if one or more of the supplied keys is an invalid format.
404 NOT FOUND	Returned if the region is not found.
414 REQUEST URI TOO LONG	Returned if the URI is longer than the system component can handle. Limiting the size to 2000 bytes will work for most components.
500 INTERNAL SERVER ERROR	Error encountered at VMware GemFire server. Check the HTTP response body for a stack trace of the exception.

PUT /geode/v1/{region}/{key}?op=REPLACE

Update (replace) data with key(s) if and only if the key(s) exists in region. The Key(s) must be present in the Region for the update to occur.

Resource URL

```
http://<hostname_or_http-service-bind-address>:<http-service-port>/geode/v1/{region}/
{key}?op=REPLACE
http://<hostname_or_http-service-bind-address>:<http-service-port>/geode/v1/{region}/
{key1},{key2},...{keyN}?op=REPLACE
```

Parameters

Parameter	Description	Example Values
op	When you specify REPLACE for this parameter, data is only updated if the specified key or keys are already present in the region.	REPLACE
@type	Specified in the response body. Use this to declare the domain object type of the entry value.	com.mycompany.ObjectName

Example Request

```
Request Payload: application/json
PUT //geode/v1/orders/2?op=REPLACE

Accept: application/json
Content-Type: application/json
{
     "@type":  "org.apache.geode.web.rest.domain.Order",
     "purchaseOrderNo": 1121,
     "customerId": 1012,
     "description":  "Order for XYZ Corp",
     "orderDate":  "02/10/2014",
     "deliveryDate":  "02/20/2014",
     "contact":  "Jelly Bean",
     "email":  "jelly.bean@example.com",
     "phone":  "01-2048096",
     "totalPrice": 225,
     "items": [
        {
             "itemNo": 1,
             "description":  "Product-100",
             "quantity": 12,
             "unitPrice": 5,
             "totalPrice": 60
        }
    ]
}
```

Example Success Response

```
Response Payload: null

200 OK
```

Error Codes

Status Code	Description
400 BAD REQUEST	Returned if the supplied key is not present in the region.
404 NOT FOUND	Returned if the region is not found.
500 INTERNAL SERVER ERROR	Error encountered at VMware GemFire server. Check the HTTP response body for a stack trace of the exception.

PUT /geode/v1/{region}/{key}?op=CAS

Update (compare-and-set) value having key with a new value if and only if the “@old” value sent matches the current value having key in region.

Resource URL

```
http://<hostname_or_http-service-bind-address>:<http-service-port>/geode/v1/{region}/
{key}?op=CAS
http://<hostname_or_http-service-bind-address>:<http-service-port>/geode/v1/{region}/
{key1},{key2},...{keyN}?op=CAS
```

Parameters

Parameter	Description	Example Values
op	URL parameter. When you specify CAS for this parameter, data is only updated if the @old value specified in the request body matches the existing value in the region.	CAS
@type	Specified in the response body for both the old and new value. Use this to declare the domain object type of the entry's value.	com.mycompany.ObjectName

Parameter	Description	Example Values

@old	Compare this value to the existing value in the region.	`{`
		`   "@type":  "org.apache.geode.web.rest.domain.Order",`
		`   "purchaseOrderNo": 1121,`
		`   "customerId": 1012,`
		`   "description":  "Order for  XYZ Corp",`
		`   "orderDate":  "02/10/2014",`
		`   "deliveryDate":  "02/20/2014",`
		`   "contact":  "Jelly Bean",`
		`   "email":  "jelly.bean@example.com",`
		`   "phone":  "01-2048096",`
		`   "items": [`
		`      {`
		`         "itemNo": 1,`
		`         "description": "Product-100",`
		`         "quantity": 12,`
		`         "unitPrice": 5,`
		`         "totalPrice": 60`
		`      }`
		`  ],`
		`   "totalPrice": 225`
		`}`

Parameter	Description	Example Values
@new	If @old value matches existing value, use this value to replace the existing value.	`{ "@type": "org.apache.geode.web.rest.domain.Order", "purchaseOrderNo": 1121, "customerId": 1013, "description": "Order for New Corp", "orderDate": "02/10/2014", "deliveryDate": "02/25/2014", "contact": "Vanilla Bean", "email": "vanilla.bean@example.com", "phone": "01-2048096", "items": [{ "itemNo": 12345, "description": "part 123", "quantity": 12, "unitPrice": 29.99, "totalPrice": 149.95 }], "totalPrice": 149.95 }`

Example Request

```
Request Payload: application/json

PUT /geode/v1/orders/2?op=CAS

Accept: application/json
Content-Type: application/json
{
    "@old": {
          "@type":  "org.apache.geode.web.rest.domain.Order",
          "purchaseOrderNo": 1121,
          "customerId": 1012,
          "description":  "Order for  XYZ Corp",
          "orderDate":  "02/10/2014",
          "deliveryDate":  "02/20/2014",
          "contact":  "Jelly Bean",
          "email":  "jelly.bean@example.com",
          "phone":  "01-2048096",
          "items": [
            {
                 "itemNo": 1,
                 "description":  "Product-100",
                 "quantity": 12,
                 "unitPrice": 5,
                 "totalPrice": 60
            }
        ],
         "totalPrice": 225
    },
     "@new ": {
         "@type":  "org.apache.geode.web.rest.domain.Order",
         "purchaseOrderNo": 1121,
         "customerId": 1013,
         "description":  "Order for  New Corp",
         "orderDate":  "02/10/2014",
         "deliveryDate":  "02/25/2014",
         "contact":  "Vanilla Bean",
         "email":  "vanillabean@example.com",
         "phone":  "01-2048096",
         "items": [
            {
                 "itemNo": 12345,
                 "description":  "part 123",
                 "quantity": 12,
                 "unitPrice": 29.99,
                 "totalPrice": 149.95
            }
        ],
         "totalPrice": 149.95
    }
}
```

Example Success Response

```
Response Payload: null

200 OK
```

Error Codes

Status Code	Description
400 BAD REQUEST	Returned if the supplied key is not present in the region.
404 NOT FOUND	Returned if the region is not found.
409 CONFLICT	Returned if the provided @old value of the key does not match the current value of the key.
500 INTERNAL SERVER ERROR	Error encountered at VMware GemFire server. Check the HTTP response body for a stack trace of the exception.

Example Error Response

```
Response-payload: application/json

409 Conflict
Content-Type: application/json
{
     "purchaseOrderNo": 1121,
     "customerId": 1012,
     "description":  "Order for  XYZ Corp",
     "orderDate":  "02/10/2014",
     "deliveryDate":  "02/20/2014",
     "contact":  "Jelly Bean",
     "email":  "jelly.bean@example.com",
     "phone":  "01-2048096",
     "items": [
        {
             "itemNo": 1,
             "description":  "Product-100",
             "quantity": 12,
             "unitPrice": 5,
             "totalPrice": 60
        }
    ],
     "totalPrice": 225
}
```

Implementation Notes

If the “@old” value sent by the client in the HTTP request, along with the “@new” value, does not match the existing value having key in region, then a 409 - CONFLICT error is returned indicating the mismatch in expected state. The “@old” and current value must match in order for the key to be assigned the “@new” value.

If a “CONFLICT” occurs, it is a simple matter for the client to issue a HTTP GET request for the Key (GET /geode/v1/orders/222) to get a updated copy of the value. CAS is similar to optimistic locking (as opposed to optimistic locking assuming the value will change between the time a client requests a value and subsequently updates the value) in that it assumes the client’s state is up-to-date when the client tries to update, but if not then fail, hence the 409 - CONFLICT.

DELETE /geode/v1/{region}

Delete all entries in the region. Limited to replicated regions only; not available for partitioned regions.

Resource URL

```
http://<hostname_or_http-service-bind-address>:<http-service-port>/geode/v1/{region}
```

Parameters

None.

Example Request

```
Request Payload: null
DELETE /geode/v1/orders
Accept: application/json
```

Example Success Response

```
Response Payload: null
200 OK
```

Error Codes

Status Code	Description
404 NOT FOUND	Returned if the region is not found.
500 INTERNAL SERVER ERROR	Error encountered at VMware GemFire server. Check the HTTP response body for a stack trace of the exception.

DELETE /geode/v1/{region}/{key}

Delete entry for specified key in the region.

Resource URL

```
http://<hostname_or_http-service-bind-address>:<http-service-port>/geode/v1/{region}/
{key}
```

Parameters

None.

Example Request

```
Request Payload: null
DELETE /geode/v1/orders/11
Accept: application/json
```

Example Success Response

```
Response Payload: null
200 OK
```

Error Codes

Status Code	Description
404 NOT FOUND	Returned if the region or specified key is not found.
500 INTERNAL SERVER ERROR	Error encountered at VMware GemFire server. Check the HTTP response body for a stack trace of the exception.

DELETE /geode/v1/{region}/{key1},{key2},...{keyN}

Delete entries for multiple keys in the region.

Resource URL

```
http://<hostname_or_http-service-bind-address>:<http-service-port>/geode/v1/{region}/
{key1},{key2},...{keyN}
```

Parameters

None.

Example Request

```
Request Payload: null
DELETE /geode/v1/orders/12,13
Accept: application/json
```

Example Success Response

```
Response Payload: null
200 OK
```

Error Codes

Status Code	Description
404 NOT FOUND	Returned if either the region or one or more of the specified keys is not found.
500 INTERNAL SERVER ERROR	Error encountered at VMware GemFire server. Check the HTTP response body for a stack trace of the exception.

Query Endpoints

VMware GemFire uses a query syntax based on OQL (Object Query Language) to query region data. Since VMware GemFire regions are key-value stores, values can range from simple byte arrays to complex nested objects.

- **GET /geode/v1/queries**

  List all parameterized queries by ID or name.

- **POST /geode/v1/queries?id=<queryId>&q=<OQL-statement>**

  Create (prepare) the specified parameterized query and assign the corresponding ID for lookup.

- **POST /geode/v1/queries/{queryId}**

  Execute the specified named query passing in scalar values for query parameters in the POST body.

- **PUT /geode/v1/queries/{queryId}**

  Update a named, parameterized query.

- **DELETE /geode/v1/queries/{queryId}**

  Delete the specified named query.

- **GET /geode/v1/queries/adhoc?q=<OQL-statement>**

  Run an unnamed (unidentified), ad-hoc query passed as a URL parameter.

GET /geode/v1/queries

List all parameterized queries by ID or name.

Resource URL

```
http://<hostname_or_http-service-bind-address>:<http-service-port>/geode/v1/queries
```

Parameters

None

Example Request

```
Request Payload: null
GET /geode/v1/queries/
Accept: application/json
```

Example Response

```

Response Payload: application/json

200 OK
Content-Type: application/json
Location: http://localhost:8080/geode/v1/queries

{
    "queries": [
        {
             "id":  "selectCustomer",
             "oql":  "SELECT c FROM /customers c WHERE c.customerId = $1"
        },
        {
             "id":  "selectHighRollers",
             "oql":  "SELECT DISTINCT c FROM /customers c, /orders o WHERE o.totalpric
e > $1 AND c.customerId = o.customerId"
        }
    ]
}
```

Error Codes

Status Code	Description
401 UNAUTHORIZED	Invalid Username or Password
403 FORBIDDEN	Insufficient privileges for operation
500 INTERNAL SERVER ERROR	Error encountered at VMware GemFire server. Check the HTTP response body for a stack trace of the exception.

POST /geode/v1/queries?id=<queryId>&q=<OQL-statement>

Create (prepare) the specified parameterized query and assign the corresponding ID for lookup.

Resource URL

```
http://<hostname_or_http-service-bind-address>:<http-service-port>/geode/v1/queries?id
=<queryId>&q="<OQL-statement>"
```

Parameters

Parameter	Description	Example Values
id	**Required.** Unique identifier for the named parameterized query.	selectCustomers
q	**Required.** OQL query statement. Use doublequotes to surround the OQL query statement.	`"SELECT o FROM /orders o WHERE o.quantity > $1 AND o.totalprice > $2"`

Note: For this release, you cannot specify the query string inside the request body (as JSON). You must specify the query as a URL parameter.

Example Request

```
POST /geode/v1/queries?id=selectOrders&q="SELECT o FROM /orders o WHERE o.quantity >
$1 AND o.totalprice > $2"
Accept: application/json
```

Example Success Response

```
Response Payload: null

201 CREATED
Location: http://localhost:8080/geode/v1/queries/selectOrders
```

Error Codes

Status Code	Description
400 BAD REQUEST	Query ID not specified or malformed OQL statement
401 UNAUTHORIZED	Invalid Username or Password
403 FORBIDDEN	Insufficient privileges for operation
409 CONFLICT	QueryId already assigned to another query
500 INTERNAL SERVER ERROR	Error encountered at VMware GemFire server. Check the HTTP response body for a stack trace of the exception. • Query store does not exist!

POST /geode/v1/queries/{queryId}

Execute the specified named query passing in scalar values for query parameters in the POST body.

Resource URL

```
http://<hostname_or_http-service-bind-address>:<http-service-port>/geode/v1/queries/{q
ueryId}
```

Parameters

Parameter	Description	Example Values
{queryId}	QueryID for named query.	selectOrders

Parameter	Description	Example Values

query bind parameter values	Bind parameters for the query are specified in the request body (JSON).	Specify the parameter @type and @value for each bind parameter. For example, to provide values to the following query:
		`SELECT o FROM /orders o WHERE o.quantity > $1 AND o.totalprice > $2`
		You could pass in the following JSON in the request body as the bind parameters:
		`[ { "@type": "int ", "@value": 2 }, { "@type": "double ", "@value": 110.00 } ]`

Example Request

```
POST /geode/v1/queries/selectOrders
Accept: application/json
Content-Type: application/json

[
        {
             "@type":  "int ",
             "@value": 2
        },
        {
            "@type":  "double ",
             "@value": 110.00
        }
]
```

Example Success Response

```
Response Payload: application/json

200 OK
Content-Length: <#-of-bytes>
Content-Type: application/json
```

```
[
    {
        "description":  "Purchase order for  company - B",
         "totalPrice": 350,
         "purchaseOrderNo": 1112,
         "customerId": 102,
         "deliveryDate":  "Thu Feb 20 00:00:00 IST 2014",
         "contact":  "John Doe",
         "email":  "John.Doe@pivotal.io",
         "phone":  "01-2048096",
         "items": [
            {
                 "description":  "Product-AAAA",
                 "quantity": 10,
                 "itemNo": 1,
                 "unitPrice": 20,
                 "totalPrice": 200,
                 "type-class":  "org.apache.geode.web.rest.domain.Item"
            },
            {
                 "description":  "Product-BBB",
                 "quantity": 15,
                 "itemNo": 2,
                 "unitPrice": 10,
                 "totalPrice": 150,
                 "type-class":  "org.apache.geode.web.rest.domain.Item"
            }
        ],
         "orderDate":  "Mon Feb 10 00:00:00 IST 2014",
         "type-class":  "org.apache.geode.web.rest.domain.Order"
    },
    {...},
    {...}
}
```

Error Codes

Status Code	Description
400 BAD REQUEST	JSON document in the request body (specifying query bind params) is invalid
401 UNAUTHORIZED	Invalid Username or Password
403 FORBIDDEN	Insufficient privileges for operation
404 NOT FOUND	Query with specified ID could not be found

Status Code	Description
500 INTERNAL SERVER ERROR	Encountered error at server: • Syntax of the OQL queryString is invalid! • A function was applied to a parameter that is improper for that function! • Bind parameter is not of the expected type! • Name in the query cannot be resolved!" • The number of bound parameters does not match the number of placeholders! • Query is not permitted on this type of region! • Query execution time has exceeded max query execution time (gemfire.Cache.MAX_QUERY_EXECUTION_TIME) configured! • Data referenced in from clause is not available for querying!" • Query execution gets canceled due to low memory conditions and the resource manager critical heap percentage has been set!" • Error encountered while executing named query!"

PUT /geode/v1/queries/{queryId}

Update a named, parameterized query.

Resource URL

```
http://<hostname_or_http-service-bind-address>:<http-service-port>/geode/v1/queries/{q
ueryId}
```

Parameters

Parameter	Description	Example Values
q	OQL String	"SELECT DISTINCT from /customers WHERE lastName=$1"

Note: For this release, you cannot specify the query string inside the request body (as JSON). You must specify the query as a URL parameter.

Example Request

```
PUT /geode/v1/queries/selectOrders?q="SELECT DISTINCT from /customers where lastName=
$1"
```

```
Accept: application/json
Content-Length: <#-of-bytes>
```

Example Success Response

```
Response Payload:  null

200 OK
```

Error Codes

Status Code	Description
401 UNAUTHORIZED	Invalid Username or Password
403 FORBIDDEN	Insufficient privileges for operation
404 NOT FOUND	QueryID does not exist
500 INTERNAL SERVER ERROR	Error encountered at VMware GemFire server. Check the HTTP response body for a stack trace of the exception.

Implementation Notes

This operation is idempotent, meaning multiple identical requests should have the same effect as the initial request.

DELETE /geode/v1/queries/{queryId}

Delete the specified named query.

Resource URL

```
http://<hostname_or_http-service-bind-address>:<http-service-port>/geode/v1/queries/{q
uery}
```

Parameters

Parameter	Description	Example Values
{queryId}	QueryID for named query to delete.	selectOrders

Example Request

```
DELETE /geode/v1/queries/selectOrders

Accept: application/json
Content-Type: application/json
```

Example Success Response

```
Response Payload: application/json

200 OK
```

Error Codes

Status Code	Description
401 UNAUTHORIZED	Invalid Username or Password
403 FORBIDDEN	Insufficient privileges for operation
404 NOT FOUND	Query with specified ID could not be found
500 INTERNAL SERVER ERROR	Encountered error at server. Check the HTTP response body for a stack trace of the exception.

GET /geode/v1/queries/adhoc?q=<OQL-statement>

Run an unnamed (unidentified), ad-hoc query passed as a URL parameter.

Resource URL

```
http://<hostname_or_http-service-bind-address>:<http-service-port>/geode/v1/queries/adhoc?q=<OQL-statement>
```

Parameters

Parameter	Description	Example Values
q	**Required.** OQL query statement. **Note:** Since the query string is passed in the URL, the OQL must be URL-encoded. Some HTTP clients such as Web browsers will automatically encode URLs; however, if you are not using one of those clients, you will need to URL encode the query string yourself.	`SELECT o FROM /orders o WHERE o.quantity > 2 AND o.totalprice > 110.00` (or URL encoded: `SELECT%20o%20FROM%20%2Forders%20o%20WHERE%20o.quantity%20%3E%202%20AND%20o.totalprice%20%3E%20110.00`) `SELECT * FROM /customers` (or URL encoded: `SELECT%20*%20FROM%20/customers`)

Example Request

```
curl -i "http://localhost:8080/geode/v1/queries/adhoc?q=select%20*%20%20from%20/custom
ers"
```

Example Success Response

```
Response Payload: application/json

200 OK
Content-Length: <#-of-bytes>
Content-Type: application/json
[
    {
        "firstName":  "John",
         "lastName":  "Doe",
         "customerId": 101,
    },
    {
         "firstName":  "Jane",
         "lastName":  "Doe",
         "customerId": 102,
    },
    {
       ....
    }
]
```

Error Codes

Status Code	Description
401 UNAUTHORIZED	Invalid Username or Password
403 FORBIDDEN	Insufficient privileges for operation

Status Code	Description
500 INTERNAL SERVER ERROR	Error encountered at VMware GemFire server. Check the HTTP response body for a stack trace of the exception. Some possible exceptions include: • A function was applied to a parameter that is improper for that function! • Bind parameter is not of the expected type! • Name in the query cannot be resolved! • The number of bound parameters does not match the number of placeholders! • Query is not permitted on this type of region! • Query execution time is exceeded max query execution time (gemfire.Cache.MAX_QUERY_EXECUTION_TIME) configured! • Data referenced in from clause is not available for querying! • Query execution gets canceled due to low memory conditions and the resource manager critical heap percentage has been set! • Server has encountered an error while executing Adhoc query!

Function Endpoints

VMware GemFire functions allows you to write and execute server-side transactions and data operations. These may include anything ranging from initializing components or third-party services or aggregating data.

- **GET /geode/v1/functions**

  List all registered VMware GemFire functions in the cluster.

- **POST /geode/v1/functions/{functionId}**

  Execute VMware GemFire function on entire cluster or on a specified region, members and member groups.

GET /geode/v1/functions

List all registered VMware GemFire functions in the cluster.

Resource URL

```
http://<hostname_or_http-service-bind-address>:<http-service-port>/geode/v1/functions
```

Parameters

None.

Example Request

```
GET /geode/v1/functions
Accept: application/json
```

Example Success Response

```
Response Payload: application/json

200 OK
Content-Length: <#-of-bytes>
Content-Type: application/json
Location: https://localhost:8080/geode/v1/functions

{
    "functions": [
        "AddFreeItemToOrders",
        "GetRegions",
        "GetDeliveredOrders"
    ]
}
```

Error Codes

Status Code	Description
404 NOT FOUND	Returned if no functions are found in the cluster.
500 INTERNAL SERVER ERROR	Error encountered at VMware GemFire server. Check the HTTP response body for a stack trace of the exception.

POST /geode/v1/functions/{functionId}

Execute VMware GemFire function on an entire cluster or on a specified region, members, and member groups.

Resource URL

```
/geode/v1/functions/{functionId}?[&onRegion=regionname|&onMembers=member1,member2,...,
memberN|&onGroups=group1,group2,...,groupN]
```

Parameters

- **{functionId}** This required parameter is the name of the function to execute. Place it in the resource URL, as in the example request: `AddFreeItemToOrders`.
- **onRegion** This optional parameter specifies the target region for the function. You can only invoke a function on a single region. Substitute the region's name for `regionname` within the

sample syntax `onRegion=regionname`.

- **onMembers** This optional parameter specifies the target members of the function. For multiple members, specify a comma-delimited list of member names, as in the sample `onMembers=member1,member2`.
- **onGroups** This optional parameter specifies the target groups of the function. For multiple groups, specify a comma-delimited list of group names, as in the sample `onGroups=membergroup1,membergroup2`.
- **filter** This optional parameter can only be used with the **onRegion** parameter, where the region has a `data-policy` of `PARTITION`. The parameter specifies a list of applicable keys that the function needs to filter on. There are three keys in the example resource URL:

```
http://serverURL/functions/SampleFunction?onRegion=TestPartitionRegion&filter=k
ey1,key2,key3
```

- Any *function arguments* are passed in the request body in JSON format. The content of the arguments depends on how the function is defined. Each function argument must be written as a JSON object (enclosed in braces `{ }`) and using `@type` to declare its type. `@value` can be used to specify a scalar value. Nested JSON objects (`{ }`) or JSON collections (`[ ]`) are not accepted. Collections of Java objects cannot be specified as parameters in JSON format, so if a function expects a collection of objects as one of its arguments (for example, `function(List<Item> list)`), it cannot be called using the VMware GemFire REST API. Following are some examples of functions and their arguments in JSON format:

Function signature	Function arguments in JSON format
`myFunction1(int n)`	`[ { "@type": "integer", "@value": 10 } ]`

Function signature	Function arguments in JSON format

`myFunction2(double d, Item i)`	
	```
	[
	    {
	        "@type":  "double",
	        "@value": 210
	    },
	    {
	        "@type":  "org.apache.geode.web.
	rest.domain.Item",
	        "itemNo":  "599",
	        "description":  "Part X Free on
	Bumper Offer",
	        "quantity":  "2",
	        "unitprice":  "5",
	        "totalprice":  "10.00"
	    }
	]
	```
	 Notice how \`item\` fields are written in JSON format due to the restrictions against nested objects. The following approach would be wrong:
	```
	[
	    {
	        "@type":  "double",
	        "@value": 210
	    },
	    {
	        "@type":  "org.apache.geode.web.
	rest.domain.Item",
	        "itemNo" : {
	            "@type":"double",
	            "@value": 599
	        },
	        "description": "Part X Free on B
	umper Offer",
	        "quantity":  "2",
	        "unitprice":  "5",
	        "totalprice":  "10.00"
	    }
	]
	```

Example Requests

```
Request Payload: application/json
POST /geode/v1/functions/AddFreeItemToOrders
Accept: application/json
Content-Type: application/json

[
        {
             "@type":  "double",
             "@value": 210
        },
        {
             "@type":  "org.apache.geode.web.rest.domain.Item",
             "itemNo":  "599",
             "description":  "Part X Free on Bumper Offer",
             "quantity":  "2",
             "unitprice":  "5",
             "totalprice":  "10.00"
        }
]
```

Another example:

```
Request Payload: null

POST /geode/v1/functions/getDeliveredOrders
Accept: application/json
```

Example Success Responses

```
Response Payload: null
200 OK
Location:http: //localhost:8080/geode/v1/functions/AddFreeItemToOrders
```

Another example response:

```
Response Payload: application/json

200 OK
Content-Length: 316
Content-Type: application/json
Location: http://localhost:8080/geode/v1/functions/getDeliveredOrders
[
    {
         "purchaseOrderNo":  "1121",
         "deliveryDate":  "Thu Oct 10 00:00:00 IST 2019"
    },
    {
         "purchaseOrderNo":  "777",
         "deliveryDate":  "Thu Oct 10 00:00:00 IST 2019"
    },
    {
        ...
    }
]
```

Error Codes

Status code 500 INTERNAL SERVER ERROR is an error encountered in a server. Check the HTTP response body for a stack trace of the exception.

Administrative Endpoints

Administrative endpoints provide management and monitoring functionality for the REST API interface.

- [HEAD | GET] /geode/v1/ping

  Mechanism to check for REST API server and service availability.
- GET /geode/v1/servers

  Mechanism to obtain a list of all members in the cluster that are running the REST API service.

\[HEAD | GET\] /geode/v1/ping

Mechanism to check for REST API server and service availability.

Resource URL

```
http://<hostname_or_http-service-bind-address>:<http-service-port>/geode/v1/ping
```

Parameters

None.

Example Request

```
HEAD /gemfire/v1/ping
GET /gemfire/v1/ping
```

Example Success Response

```
200 OK
```

Error Codes

Status Code	Description
404 NOT FOUND	The Developer REST API service is not available.
500 INTERNAL SERVER ERROR	Encountered error at server. Check the VMware GemFire exception trace.

GET /geode/v1/servers

Mechanism to obtain a list of all members in the cluster that are running the REST API service.

Resource URL

```
http://<hostname_or_http-service-bind-address>:<http-service-port>/geode/v1/servers
```

Parameters

None.

Example Request

```
Request Payload: null
Request: GET /geode/v1/servers
Response Payload: application/json
```

Example Success Response

```
200 OK

Content-Length: <#-of-bytes>
Content-Type: application/json; charset=utf-8

[
 "http://<HOST_NAME1>:<PORT1>",
 "http://<HOST_NAME2>:<PORT2>",
 "http://<HOST_NAME3>:<PORT3>",
 "http://<HOST_NAME4>:<PORT4>"
]
```

Error Codes

Status Code	Description
500 INTERNAL SERVER ERROR	Returned if VMware GemFire throws an error while executing the request.

Tools and Modules

Tools and Modules describes tools and modules associated with VMware GemFire.

- **gfsh (VMware GemFire SHell)**

  gfsh (pronounced "jee-fish") provides a single, powerful command-line interface from which you can launch, manage, and monitor VMware GemFire processes, data, and applications.

- **Gemcached**

  Gemcached is a VMware GemFire adapter that allows Memcached clients to communicate with a VMware GemFire server cluster, as if the servers were memcached servers. Memcached is an open-source caching solution that uses a distributed, in-memory hash map to store key-value pairs of string or object data.

- **HTTP Session Management Modules**

  The VMware GemFire HTTP Session Management modules provide fast, scalable, and reliable session replication for HTTP servers without requiring application changes.

- **Apache Lucene® Integration**

  The Apache Lucene® integration enables users to create Lucene indexes and execute Lucene searches on data stored in VMware GemFire.

- **Micrometer**

  Micrometer is a simple facade for application monitoring (as SLF4J, for example, is a simple facade for logging) which allows publishing metrics from VMware GemFire to various Application Performance Monitoring (APM) tools.

- **Tanzu Observability by Wavefront**

  The Tanzu GemFire metrics module provides out-of-the-box integration with Tanzu Observability by Wavefront.

- **Pulse**

  Pulse is a Web Application that provides a graphical dashboard for monitoring vital, real-time health and performance of VMware GemFire clusters, members, and regions.

- **Visual Statistics Display (VSD)**

  The Visual Statistics Display utility reads VMware GemFire statistics and produces graphical displays for analysis.

gfsh

`gfsh` (pronounced "jee-fish") provides a single, powerful command-line interface from which you can launch, manage, and monitor VMware GemFire processes, data, and applications.

- **What You Can Do with gfsh**

  `gfsh` supports the administration, debugging, and deployment of VMware GemFire processes and applications.

- **Starting gfsh**

  Before you start `gfsh`, confirm that you have set JAVA_HOME and that your PATH variable includes the `gfsh` executable.

- **Configuring the gfsh Environment**

  The `gfsh.bat` and `gfsh` bash script automatically append the required VMware GemFire and JDK .jar libraries to your existing CLASSPATH. There are user-configurable properties you can set for security, environment variables, logging, and troubleshooting.

- **Useful gfsh Shell Variables**

  You can use the built-in `gfsh` shell variables in scripts.

- **Basic Shell Features and Command-Line Usage**

  The `gfsh` utility provides useful features for a shell environment, including command auto-complete, preserved command history, and delimiting of multi-line commands. Context-sensitive help is available by command and by topic.

- **Tutorial—Performing Common Tasks with gfsh**

  This topic takes you through a typical sequence of tasks that you execute after starting `gfsh`.

- **Quick Reference of gfsh Commands by Functional Area**

  This quick reference sorts all commands into functional areas.

- **gfsh Command Help**

  This section provides help and usage information on all `gfsh` commands, listed alphabetically.

- **Creating and Running gfsh Command Scripts**

  `gfsh` offers several ways to run commands in a scripting environment.

- **Running gfsh Commands on the OS Command Line**

- **Mapping of cache.xml Elements to gfsh Configuration Commands.**

  You can configure a VMware GemFire cluster using either cache.xml files, or you can use `gfsh` and the cluster configuration service to configure a cluster. This section maps cache.xml elements to the `gfsh` commands that configure and manage a cluster.

What You Can Do with gfsh

`gfsh` supports the administration, debugging, and deployment of VMware GemFire processes and applications.

With `gfsh`, you can:

- Start and stop VMware GemFire processes, such as locators and cache servers

- Start and stop gateway sender and gateway receiver processes
- Deploy applications
- Create and destroy regions
- Execute functions
- Manage disk stores
- Import and export data
- Monitor VMware GemFire processes
- Launch VMware GemFire monitoring tools

The `gfsh` command line interface lets developers spend less time configuring cache instance XML, properties, logs, and statistics. gfsh commands generate reports; capture cluster-wide statistics; and support the export of statistics, logs, and configurations. Like Spring Roo, gfsh features command completion (so you do not have to know the syntax), context-sensitive help, scripting, and the ability to invoke any commands from within an application by using a simple API. The gfsh interface uses JMX/RMI to communicate with VMware GemFire processes.

You can connect gfsh to a remote cluster using the HTTP protocol. See Using gfsh to Manage a Remote Cluster Over HTTP or HTTPS.

By default, the cluster configuration service saves the configuration of your VMware GemFire cluster as you create VMware GemFire objects using gfsh. You can export this configuration and import it into another VMware GemFire cluster. See Overview of the Cluster Configuration Service.

Starting gfsh

Before you start gfsh, confirm that you have set JAVA_HOME and that your PATH variable includes the gfsh executable.

Note: On Windows, you must have the JAVA_HOME environment variable set properly to use start, stop and status commands for both locators and servers.

To launch the gfsh command-line interface, execute the following command at the prompt on any machine that is currently installed with VMware GemFire:

Start gfsh on Windows:

```
<product_directory>\bin\gfsh.bat
```

where <*product_directory*> corresponds to the location where you installed VMware GemFire.

Start gfsh on Unix:

```
<product_directory>/bin/gfsh
```

where <*product_directory*> corresponds to the location where you installed VMware GemFire. Upon execution, the `gfsh` script appends the required VMware GemFire and JDK Jar libraries to your existing CLASSPATH.

If you have successfully started `gfsh`, the `gfsh` splash screen and prompt appears.

```
c:{{{ vars.product_name }}}\Latest>gfsh.bat
    _________________________     __
   / _____/ ______/ ______/ /____/ /
  / /  __/ /___  /_____  / _____  /
 / /__/ / ____/  _____/ / /    / /
/______/_/      /______/_/    /_/

Monitor and Manage VMware GemFire
gfsh>
```

You can also run some gfsh commands directly within your terminal without entering a `gfsh` prompt. For example, on Unix/Linux you could enter:

```
$ gfsh start server --name=server1
```

or on Windows:

```
prompt> gfsh start server --name=server1
```

See Creating and Running gfsh Command Scripts for more information.

Configuring the gfsh Environment

The `gfsh.bat` and `gfsh` bash script automatically append the required VMware GemFire and JDK .jar libraries to your existing CLASSPATH. There are user-configurable properties you can set for security, environment variables, logging, and troubleshooting.

JAR Libraries in CLASSPATH

When you start up `gfsh`, it will automatically load required JAR files that have been packaged in the `gfsh-dependencies.jar` file. You do not need to modify your CLASSPATH to run `gfsh`.

The JAR files are packaged within your installation directory in the `lib` directory.

Machine Hostname

On some operating systems, you may need to ensure that the hostname of your machine is configured in your system hosts file. For example, on macOS you may need to map your machine's hostname to your IP address in the `/etc/hosts` file in order for `gfsh` and Pulse to operate correctly.

Configuring gfsh Security

Since `gfsh` must connect to a JMX Manager member to run certain commands (namely those commands that manage and monitor other members), JMX Manager configuration properties can affect `gfsh` security. In `gemfire.properties`, the following VMware GemFire properties can affect `gfsh` connection settings to the JMX Manager:

- `jmx-manager-ssl`
- `jmx-manager-port`
- `jmx-manager-password-file`

- `jmx-manager-access-file`

You may also need to verify that the ports are available and open to client connections. See Configuring a JMX Manager for details on these security properties.

Configuring gfsh Environment Variables

In addition, you can set gfsh-specific preset SHELL variables by using the `set variable` command. For example, you can set `gfsh` to run in quiet mode. Not all `gfsh` variables are modifiable. User-configurable variables include:

- APP_FETCH_SIZE
- APP_QUIET_EXECUTION

See Useful gfsh Shell Variables for more information.

Configuring gfsh Session Logging

By default, `gfsh` session logging is disabled. To enable gfsh logging, you must set the Java system property `-Dgfsh.log-level=desired_log_level` where *desired_log _level* is one of the following values: severe, warning, info, config, fine, finer, finest. For example, in Linux:

```
export JAVA_ARGS=-Dgfsh.log-level=info
```

Then, start `gfsh`.

gfsh produces a log file named **gfsh-*%u_%g*.log**. This log file records the events of an individual gfsh session. It includes environment information, such as Java and system information, and detailed command execution. The variables are replaced as follows:

- **%u** - a unique number to resolve conflicts
- **%g** - the generation number to distinguish rotated logs

gfsh uses the JDK Logger to generate gfsh session log files. See http://docs.oracle.com/javase/7/docs/api/java/util/logging/FileHandler.html for a description of how the variables are used in naming the log file. The default name of the generated gfsh log file cannot be changed.

By default, the log file is written to the current working directory where you have executed the `gfsh` or `gfsh.bat` script. To modify the directory location where log files are written, use the `gfsh.log-dir` Java system property. For example:

```
export JAVA_ARGS="-Dgfsh.log-level=info -Dgfsh.log-dir=/machinename/logs"
```

Then, start `gfsh`.

In addition, `gfsh` records a history of commands in the `${SYS_USER_HOME}/.geode/.gfsh.history` file, which you can use to create scripts or review past commands.

Member Log Files

gfsh writes several log files for any members that are started via gfsh. Useful member log files include:

- **<locator_name>.log**. Details a locator's configuration (including all gemfire.properties) and all activity that occurs on the locator after startup. This log file is written to a directory that is named after the locator. For example, if you start a locator named locator1, the file is written as `locator1.log` in the `<product_dir>/locator1` directory.
- **vf.gf.locator.pid**. Contains the process ID of the locator. You can use the PID to stop or view the status of this locator. This file is written to the same directory location as the locator's log file .
- **<server_name>.log**. Details a server's configuration (including all gemfire.properties) and all activity that occurs on the server after startup. This log file is written to a directory that is named after the server. For example, if you start a server named server1, the file is written as `server1.log` in the `<product_dir>/server1` directory. If you stop and start the server with an identical name, the older log files are kept in the same directory but renamed for versioning purposes.
- **vf.gf.server.pid**. Contains the process ID of the server. You can use the PID to stop or view the status of this server. This file is written to the same location as the server log file.

Viewing Standard Output and Standard Error

By default, VMware GemFire does not show messages written by the application to standard output and standard error. To allow these messages to be written to the locator and server log files, respectively, specify the `--redirect-output` option with the gfsh `start locator` or `start server` commands. For example, the following command causes `stdout` and `stderr` messages to be written to the `locator1.log` file:

```
gfsh> start locator --name=locator1 --redirect-output
```

Tab Completion

This section applies only to UNIX installations.

When you run gfsh commands from a UNIX bash shell, you can enable automatic tab-completion in the shell by running the following command:

```
source <gemfire-install-directory>/bin/gfsh-completion.bash
```

After running this command, you can use auto completion when running gfsh commands from the bash shell.

See Using Tab Completion.

Command History and gfsh.history

A history of commands that have been executed successfully is logged in `.gfsh.history` file in a `.gemfire` directory under the home directory of the user running gfsh. You can also export a history file by using the `history --file=your_file_name` command.

JMX Manager Update Rate and System Monitoring

When you perform data operations (such as put) and then monitor the state of the system (such as using the gfsh `show metrics` command or VMware GemFire Pulse), the monitored system may not immediately reflect the most recent operations. For example, if you perform a put operation and then immediately execute the `show metrics` gfsh command, you may not see the correct number of entries in the region. The management layer updates every 2 seconds. Wait a few seconds after performing operational activity to see the most accurate results.

You can modify the `jmx-manager-update-rate` property in `gemfire.properties` to increase or decrease the rate (specified in milliseconds) at which updates are pushed to the JMX Manager. This property setting should be greater than or equal to the `statistic-sample-rate`. You may want to increase this rate if you are experiencing performance issues; however, setting this value too high will cause stale values to be seen in `gfsh` and VMware GemFire Pulse.

Formatting of Results

This section applies only to UNIX installations.

gfsh commands such as `query` produce output with wide columns that may become misaligned and require manual reformatting to view the output. If the output cannot fit in the available width of the terminal, gfsh automatically trims the columns widths to fit. You can disable this behavior by setting the gfsh environment variable `APP_RESULT_VIEWER` to an arbitrary value that is not `external` or `basic`.

You can set the `APP_RESULT_VIEWER` variable to `external` to enable viewing of the output using the UNIX `less` command.

See Configuring gfsh Environment Variables.

Useful gfsh Shell Variables

You can use the built-in `gfsh` shell variables in scripts.

You can also use the `set variable` command to modify shell behavior or to define your own variables.

To see a list of all gfsh shell variables and their current values, use the following command:

```
gfsh>echo --string=$*
```

To obtain the current value of an existing variable, use the following command syntax (the variable must be enclosed in braces):

```
gfsh>echo --string=${VARIABLE}
```

For example:

```
gfsh>echo --string=${SYS_CLASSPATH}
```

System Variables

SYS_CLASSPATH	CLASSPATH of the gfsh JVM (read only).
SYS_GEMFIRE_DIR	Product directory where VMware GemFire has been installed (read only).
SYS_HOST_NAME	Host from which gfsh is started (read only).
SYS_JAVA_VERSION	Java version used (read only).
SYS_OS	OS name (read only).
SYS_OS_LINE_SEPARATOR	Line separator (\ or ^) variable that you can use when writing gfsh scripts. (read only).
SYS_USER	User name (read only).
SYS_USER_HOME	User's home directory (read only).

GFSH Environment Variables

APP_FETCH_SIZE	Fetch size to be used while querying. Values: 0 - 2147483647. Default value is 100.
APP_LAST_EXIT_STATUS	Last command exit status. Similar to $? (Unix) and %errorlevel% (Windows). Values: 0 (successful), 1 (error), 2(crash) (read only).
APP_LOGGING_ENABLED	Whether gfsh logging is enabled. Default: false (read only). You can enable gfsh logging by setting the `gfsh.log-level` Java system property to a supported Java log level.
APP_LOG_FILE	Path and name of current gfsh log file (read only).
APP_NAME	Name of the application– "gfsh" (read only).
APP_PWD	Current working directory where gfsh was launched (read only).
APP_QUERY_RESULTS_DISPLAY_MODE	Toggle the display mode for returning query results. Values: table or catalog. Default value is table.
APP_QUIET_EXECUTION	Whether the execution should be in quiet mode. Values (case insensitive): true, false. Default value is false.
APP_RESULT_VIEWER	Unix only. Set this variable to `external` to enable viewing of the output using the UNIX `less` command. Default value is basic (gfsh).

Basic Shell Features and Command-Line Usage

The `gfsh` utility provides useful features for a shell environment, including command auto-complete, preserved command history, and delimiting of multi-line commands. Context-sensitive help is available by command and by topic.

To view a list of available gfsh commands, press Tab at an empty prompt.

The list of commands you see depends on whether you are connected to a VMware GemFire cluster. If you are not connected, you see a list of local commands that are available.

Use the hint command to get information on a particular topic.

The hint command displays a one-line description and associated commands for a specified topic. For example, **hint data** returns a description of the data topic and all the possible actions available for data:

```
gfsh>hint Data
User data as stored in regions of the VMware GemFire distributed system.

clear defined indexes: Clears all the defined indexes.
create defined indexes: Creates all the defined indexes.
create index: Create an index that can be used when executing queries.
create lucene index: Create a lucene index that can be used to execute queries.
define index: Define an index that can be used when executing queries.
describe lucene index: Display the description of lucene indexes created for all membe
rs.
destroy index: Destroy/Remove the specified index.
destroy lucene index: Destroy the lucene index.
export data: Export user data from a region to a file.
get: Display an entry in a region. If using a region whose key and value classes have
been set, then specifying --key-class and --value-class is unnecessary.
import data: Import user data from a file to a region.
list indexes: Display the list of indexes created for all members.
list lucene indexes: Display the list of lucene indexes created for all members.
locate entry: Identifies the location, including host, member and region, of entries t
hat have the specified key.
put: Add/Update an entry in a region. If using a region whose key and value classes ha
ve been set, then specifying --key-class and --value-class is unnecessary.
rebalance: Rebalance partitioned regions. The default is for all partitioned regions t
o be rebalanced.
remove: Remove an entry from a region. If using a region whose key class has been set,
then specifying --key-class is unnecessary.
search lucene: Search lucene index
```

To view a list of hint topics, type `hint`.

Use the help command to get information on a particular command.

Depending on the command, typing `help <command-name>` displays usage information for that particular command or a list of commands related to the command.

For example, type `help start` to display a list of start commands with short descriptions. The descriptions indicate whether the command is available, and depend on the connection status of `gfsh`. The following example assumes `gfsh` is not currently connected (via connection to a JMX Manager node), so some start commands are unavailable.

```
gfsh>help start
start gateway-receiver (Not Available)
    Start the Gateway Receiver on a member or members.
start gateway-sender (Not Available)
    Start the Gateway Sender on a member or members.
start jconsole (Available)
    Start the JDK's JConsole tool in a separate process. JConsole will be
    launched, but connecting to GemFire must be done manually.
start jvisualvm (Available)
    Start the JDK's Java VisualVM (jvisualvm) tool in a separate process. Java
    VisualVM will be launched, but connecting to GemFire must be done manually.
start locator (Available)
    Start a Locator.
start pulse (Available)
    Open a new window in the default Web browser with the URL for the Pulse
    application.
start server (Available)
    Start a GemFire Cache Server.
```

Use the Tab key to auto-complete a command or trigger possible completions.

Use the Tab key after entering a partial command to trigger auto-completion of the command or a list of possible command completions. For example, hitting Tab after typing `hint` displays all available topics:

```
gfsh>hint

Configuration           Data                    Debug-Utility
Disk Store              Function Execution      GFSH
GemFire                 Help                    JMX
Lifecycle               Locator                 Management-Monitoring
Manager                 Region                  Server
Statistics
```

Hitting tab after typing `hint d` displays the topics available that begin with 'd':

```
gfsh>hint d

data            debug-Utility   disk Store
```

Auto-completion also supplies available parameters and arguments to commands; for example, hitting tab after typing `start` will list all the processes that can be started.

```
gfsh>start

start data-browser   start jconsole       start jvisualvm
start locator        start pulse          start server
```

Hitting tab after `start locator` will populate the `--name` parameter after the command. (If you do not specify a member name, gfsh will automatically pick a random name. This is useful for automation.)

Access command history with the Up arrow.

Access a command in your shell history by scrolling through previous commands with the Up arrow.

Delimit multi-line commands with a backslash.

When entering long commands, you can break the command line up by using the backslash character ('') as a delimiter. For example:

```
gfsh>create region --name=region1 \
--type=REPLICATE_PERSISTENT \
--cache-writer=org.apache.geode.examples.MyCacheWriter \
--group=Group1 --disk-store=DiskStore1
```

Enclose strings that contain spaces or commas with single or double quotes

When you execute gfsh commands in the gfsh command shell, enclose any strings that contain spaces or commas in single quotes. For example:

```
gfsh>start locator --name='locator 1'
```

```
start locator --name=locator1 --port=9009 --mcast-port=0\
 --J='-Dgemfire.remote-locators=192.0.2.0[9009],192.0.2.1[9009]'
```

When you execute multiple gfsh commands in a single line from the operating system shell, enclose the gfsh commands in double quotes. Within the double quotes, enclose any strings that contain spaces or commas with single quotes. For example:

```
$ gfsh -e "start locator --name='locator 1'" -e "start server --name=server1"
```

Tutorial—Performing Common Tasks with gfsh

This topic takes you through a typical sequence of tasks that you execute after starting `gfsh`.

Step 1: Create a scratch working directory and change to that directory. For example:

```
$ mkdir gfsh_tutorial
$ cd gfsh_tutorial
```

Step 1: Start a gfsh prompt.

```
$ gfsh

    _________________________     __
   / _____/ ______/ ______/ /____/ /
  / /  __/ /___  /_____  / _____  /
 / /__/ / ____/  _____/ / /    / /
/______/_/      /______/_/    /_/


Monitor and Manage VMware GemFire
gfsh>
```

See Starting gfsh for details.

Step 2: Start up a locator. Enter the following command:

```
gfsh>start locator --name=locator1
```

The following output appears:

```
gfsh>start locator --name=locator1
.....
Locator in /home/username/gfsh_tutorial/locator1 on 192.0.2.0[10334]
as locator1 is currently online.
Process ID: 67666
Uptime: 6 seconds
VMware GemFire Version: 9.10
Java Version: 1..0_272
Log File: /home/username/gfsh_tutorial/locator1.log
JVM Arguments: -Dgemfire.enable-cluster-configuration=true
 -Dgemfire.load-cluster-configuration-from-dir=false
 -Dgemfire.launcher.registerSignalHandlers=true
 -Djava.awt.headless=true -Dsun.rmi.dgc.server.gcInterval=9223372036854775806
Class-Path: /home/username/geode/geode-assembly/build/install/apache-geode/lib
/geode-core-1.2.0.jar:/home/username/geode/geode-assembly/build/install/apache-geode
/lib/geode-dependencies.jar

Successfully connected to: JMX Manager [host=192.0.2.0, port=1099]
```

```
Cluster configuration service is up and running.
```

If you run `start locator` from gfsh without specifying the member name, gfsh will automatically pick a random member name. This is useful for automation.

In your file system, examine the folder location where you executed `gfsh`. Notice that the `start locator` command has automatically created a working directory (using the name of the locator), and within that working directory, it has created a log file, a status file, and a .pid (containing the locator's process ID) for this locator.

In addition, because no other JMX Manager exists yet, notice that `gfsh` has automatically started an embedded JMX Manager on port 1099 within the locator and has connected you to that JMX Manager.

Step 3: Examine the existing gfsh connection.

In the current shell, type the following command:

```
gfsh>describe connection
```

If you are connected to the JMX Manager started within the locator that you started in Step 2, the following output appears:

```
gfsh>describe connection
Connection Endpoints
--------------------
ubuntu.local[1099]
```

Notice that the JMX Manager is on 1099 whereas the locator was assigned the default port of 10334.

Step 4: Connect to the same locator/JMX Manager from a different terminal.

This step shows you how to connect to a locator/JMX Manager. Open a second terminal window, and start a second `gfsh` prompt. Type the same command as you did in Step 3 in the second prompt:

```
gfsh>describe connection
```

This time, notice that you are not connected to a JMX Manager, and the following output appears:

```
gfsh>describe connection
Connection Endpoints
--------------------
Not connected
```

Type the following command in the second `gfsh` terminal:

```
gfsh>connect
```

The command will connect you to the currently running local locator that you started in Step 2.

```
gfsh>connect
Connecting to Locator at [host=localhost, port=10334] ..
Connecting to Manager at [host=ubuntu.local, port=1099] ..
Successfully connected to: [host=ubuntu.local, port=1099]
```

Note that if you had used a custom `--port` when starting your locator, or you were connecting from the `gfsh` prompt on another member, you would also need to specify `--locator=hostname[port]` when connecting to the cluster. For example (type `disconnect` first if you want to try this next command):

```
gfsh>connect --locator=localhost[10334]
Connecting to Locator at [host=localhost, port=10334] ..
Connecting to Manager at [host=ubuntu.local, port=1099] ..
Successfully connected to: [host=ubuntu.local, port=1099]
```

Another way to connect `gfsh` to the cluster would be to connect to directly to the JMX Manager running inside the locator. For example (type `disconnect` first if you want to try this next command):

```
gfsh>connect --jmx-manager=localhost[1099]
Connecting to Manager at [host=localhost, port=1099] ..
Successfully connected to: [host=localhost, port=1099]
```

In addition, you can connect to remote clusters over HTTP. See Using gfsh to Manage a Remote Cluster Over HTTP or HTTPS.

Step 5: Disconnect and close the second terminal window. Type the following commands to disconnect and exit the second `gfsh` prompt:

```
gfsh>disconnect
Disconnecting from: localhost[1099]
Disconnected from : localhost[1099]
gfsh>exit
```

Close the second terminal window.

Step 6: Start up a server. Return to your first terminal window, and start up a cache server that uses the locator you started in Step 2. Type the following command:

```
gfsh>start server --name=server1 --locators=localhost[10334]
```

If the server starts successfully, the following output appears:

```
gfsh>start server --name=server1 --locators=localhost[10334]
Starting a VMware GemFire Server in /home/username/gfsh_tutorial/server1/server1.lo
g...
...
Server in /home/username/gfsh_tutorial/server1 on 192.0.2.0[40404] as server1
is currently online.
Process ID: 68375
Uptime: 4 seconds
VMware GemFire Version: 9.10
Java Version: 1..0_272
Log File: /home/username//gfsh_tutorial/server1/server1.log
JVM Arguments: -Dgemfire.locators=localhost[10334]
 -Dgemfire.use-cluster-configuration=true -Dgemfire.start-dev-rest-api=false
 -XX:OnOutOfMemoryError=kill -KILL %p
 -Dgemfire.launcher.registerSignalHandlers=true
 -Djava.awt.headless=true -Dsun.rmi.dgc.server.gcInterval=9223372036854775806
Class-Path: /home/username/geode/geode-assembly/build/install/apache-geode/lib
/geode-core-1.2.0.jar:/home/username/geode/geode-assembly/build/install
/apache-geode/lib/geode-dependencies.jar
```

If you run `start server` from gfsh without specifying the member name, gfsh will automatically pick a random member name. This is useful for automation.

In your file system, examine the folder location where you executed `gfsh`. Notice that just like the `start locator` command, the `start server` command has automatically created a working directory (named after the server), and within that working directory, it has created a log file and a .pid (containing the server's process ID) for this cache server. In addition, it has also written log files.

Step 7: List members. Use the `list members` command to view the current members of the cluster you have just created.

```
gfsh>list members
  Name       | Id
------------ | ---------------------------------------
Coordinator: | ubuntu(locator1:5610:locator)<ec><v0>:34168
locator1     | ubuntu(locator1:5610:locator)<ec><v0>:34168
server1      | ubuntu(server1:5931)<v1>:35285
```

Step 8: View member details by executing the `describe member` command.

```
gfsh>describe member --name=server1
Name        : server1
Id          : ubuntu(server1:5931)<v1>:35285
Host        : ubuntu.local
Regions     :
PID         : 5931
Groups      :
Used Heap   : 12M
Max Heap    : 239M
Working Dir : /home/username/gfsh_tutorial/server1
Log file    : /home/username/gfsh_tutorial/server1/server1.log
Locators    : localhost[10334]

Cache Server Information
Server Bind             :
Server Port             : 40404
Running                 : true
Client Connections      : 0
```

Note that no regions have been assigned to this member yet.

Step 9: Create your first region. Type the following command followed by the tab key:

```
gfsh>create region --name=region1 --type=
```

A list of possible region types appears, followed by the partial command you entered:

```
gfsh>create region --name=region1 --type=

PARTITION
```

```
PARTITION_REDUNDANT
PARTITION_PERSISTENT
PARTITION_REDUNDANT_PERSISTENT
PARTITION_OVERFLOW
PARTITION_REDUNDANT_OVERFLOW
PARTITION_PERSISTENT_OVERFLOW
PARTITION_REDUNDANT_PERSISTENT_OVERFLOW
PARTITION_HEAP_LRU
PARTITION_REDUNDANT_HEAP_LRU
REPLICATE
REPLICATE_PERSISTENT
REPLICATE_OVERFLOW
REPLICATE_PERSISTENT_OVERFLOW
REPLICATE_HEAP_LRU
LOCAL
LOCAL_PERSISTENT
LOCAL_HEAP_LRU
LOCAL_OVERFLOW
LOCAL_PERSISTENT_OVERFLOW
PARTITION_PROXY
PARTITION_PROXY_REDUNDANT
REPLICATE_PROXY

gfsh>create region --name=region1 --type=
```

Complete the command with the type of region you want to create. For example, create a local region:

```
gfsh>create region --name=region1 --type=LOCAL
Member  | Status
------- | --------------------------------------
server1 | Region "/region1" created on "server1"
```

Because only one server is in the cluster at the moment, the command creates the local region on server1.

Step 10: Start another server. This time specify a `--server-port` argument with a different server port because you are starting a cache server process on the same host machine.

```
gfsh>start server --name=server2 --server-port=40405
Starting a VMware GemFire Server in /home/username/gfsh_tutorial/server2...
...
Server in /home/username/gfsh_tutorial/server2 on 192.0.2.0[40405] as
server2 is currently online.
Process ID: 68423
Uptime: 4 seconds
VMware GemFire Version: 9.10
Java Version: 1..0_272
Log File: /home/username/gfsh_tutorial/server2/server2.log
JVM Arguments: -Dgemfire.default.locators=192.0.2.0[10334]
 -Dgemfire.use-cluster-configuration=true -Dgemfire.start-dev-rest-api=false
 -XX:OnOutOfMemoryError=kill -KILL %p -Dgemfire.launcher.registerSignalHandlers=true
 -Djava.awt.headless=true -Dsun.rmi.dgc.server.gcInterval=9223372036854775806
Class-Path: /home/username/geode/geode-assembly/build/install/apache-geode
/lib/geode-core-1.2.0.jar:/home/username/geode/geode-assembly/build/install
/apache-geode/lib/geode-dependencies.jar
```

Step 11: Create a replicated region.

```
gfsh>create region --name=region2 --type=REPLICATE
Member  | Status
------- | --------------------------------------
server1 | Region "/region2" created on "server1"
server2 | Region "/region2" created on "server2"
```

Step 12: Create a partitioned region.

```
gfsh>create region --name=region3 --type=PARTITION
Member  | Status
------- | --------------------------------------
server1 | Region "/region3" created on "server1"
server2 | Region "/region3" created on "server2"
```

Step 13: Create a replicated, persistent region.

```
gfsh>create region --name=region4 --type=REPLICATE_PERSISTENT
Member  | Status
------- | --------------------------------------
server1 | Region "/region4" created on "server1"
server2 | Region "/region4" created on "server2"
```

Step 14: List regions. A list of all the regions you just created displays.

```
gfsh>list regions
List of regions
---------------
region1
region2
region3
region4
```

Step 15: View member details again by executing the `describe member` command.

```
gfsh>describe member --name=server1
Name        : server1
Id          : ubuntu(server1:5931)<v1>:35285
Host        : ubuntu.local
Regions     : region4
              region3
              region2
              region1
PID         : 5931
Groups      :
Used Heap   : 14M
Max Heap    : 239M
Working Dir : /home/username/gfsh_tutorial/server1
Log file    : /home/username/gfsh_tutorial/server1/server1.log
Locators    : localhost[10334]

Cache Server Information
Server Bind              :
Server Port              : 40404
Running                  : true
Client Connections       : 0
```

Notice that all the regions that you created now appear in the “Regions” section of the member description.

```
gfsh>describe member --name=server2
Name        : server2
Id          : ubuntu(server2:6092)<v2>:17443
Host        : ubuntu.local
Regions     : region4
              region3
              region2
              region1
PID         : 6092
Groups      :
Used Heap   : 14M
Max Heap    : 239M
Working Dir : /home/username/gfsh_tutorial/server2
Log file    : /home/username/gfsh_tutorial/server2/server2.log
Locators    : 192.0.2.0[10334]

Cache Server Information
Server Bind             :
Server Port             : 40405
Running                 : true
Client Connections      : 0
```

Note that even though you brought up the second server after creating the first region (region1), the second server still lists region1 because it picked up its configuration from the cluster configuration service.

Step 16: Put data in a local region. Enter the following put command:

```
gfsh>put --key=('123') --value=('ABC') --region=region1
Result      : true
Key Class   : java.lang.String
Key         : ('123')
Value Class : java.lang.String
Old Value   : <NULL>
```

Step 17: Put data in a replicated region. Enter the following put command:

```
gfsh>put --key=('123abc') --value=('Hello World!!') --region=region2
Result      : true
Key Class   : java.lang.String
Key         : ('123abc')
Value Class : java.lang.String
Old Value   : <NULL>
```

Step 18: Retrieve data. You can use `locate entry`, `query` or `get` to return the data you just put into the region.

For example, using the `get` command:

```
gfsh>get --key=('123') --region=region1
Result      : true
Key Class   : java.lang.String
Key         : ('123')
```

```
Value Class : java.lang.String
Value       : ('ABC')
```

For example, using the `locate entry` command:

```
gfsh>locate entry --key=('123abc') --region=region2
Result          : true
Key Class       : java.lang.String
Key             : ('123abc')
Locations Found : 2


MemberName | MemberId
---------- | -------------------------------
server2    | ubuntu(server2:6092)<v2>:17443
server1    | ubuntu(server1:5931)<v1>:35285
```

Notice that because the entry was put into a replicated region, the entry is located on both cluster members.

For example, using the `query` command:

```
gfsh>query --query='SELECT * FROM /region2'

Result     : true
startCount : 0
endCount   : 20
Rows       : 1

Result
-----------------
('Hello World!!')

NEXT_STEP_NAME : END
```

Step 19: Export your data. To save region data, you can use the `export data` command.

For example:

```
gfsh>export data --region=region1 --file=region1.gfd --member=server1
```

You can later use the `import data` command to import that data into the same region on another member.

Step 20: Shutdown the cluster.

```
gfsh>shutdown --include-locators=true
```

Quick Reference of gfsh Commands by Functional Area

This quick reference sorts all commands into functional areas.

Click a command to see additional information, including syntax, a list of options, and examples.

- **Basic GemFire gfsh Commands**
- **Configuration Commands**

- Data Commands
- Deployment Commands
- Disk Store Commands
- Durable CQ and Client Commands
- Function Execution Commands
- Gateway (WAN) Commands
- GemFire Monitoring Commands
- Index Commands
- JMX Connection Commands
- Locator Commands
- PDX Commands
- Region Commands
- Server Commands

gfsh Command Help

This section provides help and usage information on all `gfsh` commands, listed alphabetically.

- **alter**

  Modify an existing VMware GemFire resource.

- **backup disk-store**

  Back up persistent data from all members to the specified directory.

- **change loglevel**

  Changes the logging level on specified members.

- **clear defined indexes**

  Clears all the defined indexes.

- **close**

  Close durable client CQs and durable clients.

- **compact**

  Compact online and offline disk-stores.

- **configure**

  Configure Portable Data eXchange for all the cache(s) in the cluster.

- **connect**

  Connect to a jmx-manager either directly or via a locator.

- **create**

Create async-event-queues, disk-stores, gateway receivers, gateway senders, indexes, and regions.

- **debug**

  Activate or deactivate debugging output in `gfsh`.

- **define index**

  Define an index that can be used when executing queries. Then, you can execute a single command to create multiple indexes all at once using `create defined indexes`.

- **deploy**

  Deploy JAR-packaged applications to a member or members.

- **describe**

  Display details of a member's configuration, shell connection, disk-stores, members, or regions.

- **destroy**

  Delete or unregister functions, remove indexes, disk stores and regions.

- **disconnect**

  Close any active connection(s).

- **echo**

  Echo the given text, which may include system and user variables.

- **execute function**

  Execute functions on members or regions.

- **exit**

  Exit the `gfsh` shell. You can also use `quit` to exit the shell.

- **export**

  Export configurations, data, logs and stack-traces.

- **gc**

  Force GC (Garbage Collection) on a member or members.

- **get**

  Display an entry in a region.

- **help**

  Display syntax and usage information for all the available commands.

- **hint**

  Display information on topics and a list of commands associated with a topic.

- **history**

  Show or save the command history.

- **import**

You can import data into a region or import an existing cluster configuration into the cluster.

- **list**

  List existing VMware GemFire resources such as deployed applications, disk-stores, functions, members, servers, and regions.

- **load-balance gateway-sender**

  Causes the specified gateway sender to close its current connections and reconnect to remote gateway receivers in a more balanced fashion.

- **locate entry**

  Locate a region entry on a member.

- **netstat**

  Report network information and statistics via the "netstat" operating system command.

- **pause gateway-sender**

  Pause a gateway sender.

- **pdx rename**

  Renames PDX types in an offline disk store.

- **put**

  Add or update a region entry.

- **query**

  Run queries against VMware GemFire regions.

- **rebalance**

  Rebalance partitioned regions.

- **remove**

  Remove an entry from a region.

- **resume gateway-sender**

  Resume any gateway senders that you have paused.

- **revoke missing-disk-store**

  Instruct the member(s) of a distributed system to stop waiting for a disk store to be available.

- **run**

  Execute a set of GFSH commands.

- **set variable**

  Set variables in the GFSH environment.

- **sh**

  Execute operating system commands.

- **show**

  Display deadlocks, logs, metrics and missing disk-stores.

- **shutdown**

  Stop all members.

- **sleep**

  Delay `gfsh` command execution.

- **start**

  Start servers, locators, gateway senders and gateway receivers, and monitoring tools.

- **status**

  Check the status of the cluster configuration service and VMware GemFire member processes, including locators, gateway receivers, gateway senders, and servers.

- **stop**

  Stop gateway receivers, gateway senders, locators and servers.

- **undeploy**

  Undeploy the JAR files that were deployed on members or groups using `deploy` command.

- **validate offline-disk-store**

  Validate offline disk stores.

- **version**

  Display product version information.

alter

Modify an existing VMware GemFire resource.

- **alter async-event-queue**

  Modifies attributes of an async event queue

- **alter disk-store**

  Modifies or removes a region from an offline disk-store.

- **alter query-service**

  Alter configuration details of the query configuration service.

- **alter region**

  Alters the configuration of a region.

- **alter runtime**

  Alters configuration properties for all members or a subset of members while the member or members are running.

alter async-event-queue

Alter attributes of a specified asynchronous event queue. Each server hosting the specified async event queue must be restarted for the new attribute settings to take effect on that server.

Availability: Online. You must be connected in gfsh to a JMX Manager member to use this command.

Syntax:

```
alter async-event-queue --id=value [--batch-size=value] [--batch-time-interval=value]
    [--max-queue-memory=value] [--if-exists(=value)] [--pause-event-processing(=valu
e)]
```

The required option, `--id`, identifies the async event queue to be altered.

Parameters, alter async-event-queue

Name	Description
--id	*Required*. ID of the async event queue to be changed
--batch-size	Maximum number of events that a batch can contain
--batch-time-interval	Maximum amount of time, in ms, that can elapse before a batch is delivered
--max-queue-memory	Maximum amount of memory, in megabytes, that the queue can consume before overflowing to disk
--if-exists	If the specified async event queue does not exist, gfsh responds with a message to that effect. If this parameter is true, the response is prefixed with the label "Skipping: ". Useful for scripted tests. Default (if the parameter is not specified): false. Default (if the parameter is specified without value): true.
--pause-event-processing	Specifies whether event dispatching from the queue to the listener(s) will be paused when the AsyncEventQueue is started. Default (if the parameter is not specified): false. Default (if the parameter is specified without value): true.

Example Commands:

```
alter async-event-queue --id=myAsyncEventQueue --batch-size=50 --if-exists
```

alter disk-store

Modify or remove a region from an offline disk-store.

When modifying a region's configuration, it is customary to take the region off-line and restart using the new configuration. You can use the `alter disk-store` command to change the configuration of the region stored in the disk-store to match the configuration you will use at restart.

Availability: Offline.

Syntax:

```
alter disk-store --name=value --region=value --disk-dirs=value(,value)*
    [--compressor(=value)] [--concurrency-level=value]
    [--enable-statistics=value] [--initial-capacity=value] [--load-factor=value]
    [--lru-algorithm=value] [--lru-action=value] [--lru-limit=value]
    [--off-heap(=value)] [--remove(=value)]
```

The three required options, `--name`, `--region`, and `--disk-dirs`, identify the disk store and region to be altered. If no additional options are specified, `gfsh` displays the current configuration without making any changes.

Parameters, alter disk-store

Name	Description
--name	*Required*. Name of the disk-store whose contents will be altered.
--region	*Required*. Name (including path) of the region using the disk store.
--disk-dirs	*Required*. Directories where the data for the disk store was previously written.
--compressor	The fully-qualified class name of the compressor to use when compressing region entry values. A value of `none` removes the compressor.
--concurrency-level	An estimate of the maximum number of application threads that will concurrently access a region entry. Together with --initial-capacity and --load-factor, sets the parameters on the underlying `java.util.ConcurrentHashMap` used for storing region entries. This attribute does not apply to partitioned regions.
--enable-statistics	Enables statistics for the region specified by the `--region` option. Valid values are true or false. If the parameter is specified without a value, the value of true is used.
--initial-capacity	Together with --concurrency-level and --load-factor, sets the parameters on the underlying `java.util.ConcurrentHashMap` used for storing region entries.
--load-factor	Together with --concurrency-level and --initial-capacity, sets the parameters on the underlying `java.util.ConcurrentHashMap` used for storing region entries. This must be a floating point number between 0 and 1, inclusive.
--lru-action	Action to take when evicting entries from the region. Valid values are: • `none` • `overflow-to-disk` • `local-destroy`

Name	Description
--lru-algorithm	Least recently used eviction algorithm. Valid types are: • `none` • `lru-entry-count` • `lru-heap-percentage` • `lru-memory-size`
--lru-limit	Number of entries allowed in the region before eviction occurs.
--off-heap	Specifies whether the region values are in heap memory or off-heap memory. When true, region values are in off-heap memory. If the parameter is specified without a value, the value of true is used.
--remove	Specifies whether to remove the region from the disk-store. If the parameter is specified without a value, the value of true is used. Note: --remove deletes all persistent data for the region. Consider copying the disk store files to a backup before using this option if you might want to retrieve the data at a later date.

Example Commands:

```
alter disk-store --name=DiskStore1 --region=region1 --disk-dirs=/Disks/DiskStore1 --of
f-heap
alter disk-store --name=DiskStore1 --region=region1 --disk-dirs=/Disks/DiskStore1 --re
move
```

alter query-service

Alter configuration details of the query configuration service.

Availability: Online. You must be connected in gfsh to a JMX Manager member to use this command.

Syntax:

```
alter query-service --method-authorizer=value
    [--authorizer-parameters=value(;value)*] [--force-update(=value)]
```

Parameters, alter query-service

Name	Description
--method-authorizer	*Required.* Fully qualified class name of the `MethodInvocationAuthorizer` to be used for query authorization.
--authorizer-parameters	A **semicolon-separated** list of parameters to be used by the specified `MethodInvocationAuthorizer`. This requires that a method-authorizer option has been specified.

Name	Description
--force-update	Specifies whether to forcibly update the `MethodInvocationAuthorizer`, even when there are continuous queries registered in the member. Default (if the parameter is not specified): `false`. Default (if the parameter is specified without value): `true`. **Note:** when set as `true`, any registered CQ will pick up the new `MethodInvocationAuthorizer` and invalidate its internal cache; consider checking that the new `MethodInvocationAuthorizer` allows the methods invoked by the CQs before using this option.

Example Commands:

```
alter query-service --method-authorizer=org.apache.geode.cache.query.security.Unrestri
ctedMethodAuthorizer
alter query-service --method-authorizer=org.apache.geode.cache.query.security.Unrestri
ctedMethodAuthorizer --force-update=true
alter query-service --method-authorizer=org.apache.geode.cache.query.security.JavaBean
AccessorMethodAuthorizer --authorizer-parameters=java.lang;java.util
```

alter region

Alters the configuration of a region.

See Specifying JSON within Command-Line Options for syntax details.

Availability: Online. You must be connected in gfsh to a JMX Manager member to use this command.

Syntax:

```
alter region --name=value [--groups=value(,value)*]
    [--entry-idle-time-expiration=value]
    [--entry-idle-time-expiration-action(=value)?]
    [--entry-time-to-live-expiration=value]
    [--entry-time-to-live-expiration-action(=value)?]
    [--entry-idle-time-custom-expiry=value] [--entry-time-to-live-custom-expiry=value]
    [--region-idle-time-expiration=value]
    [--region-idle-time-expiration-action(=value)?]
    [--region-time-to-live-expiration=value]
    [--region-time-to-live-expiration-action(=value)?]
    [--cache-listener=value(,value)*] [--cache-loader=value]
    [--cache-writer=value] [--async-event-queue-id=value(,value)*]
    [--gateway-sender-id=value(,value)*] [--enable-cloning(=value)?]
    [--eviction-max(=value)?]
```

Parameters, alter region

Name	Description	Default Value
--async-event-queue-id	IDs of the Async Event Queues that will be used for write-behind operations.	
--cache-listener	Fully qualified class name of a plug-in to be instantiated for receiving after-event notification of changes to the region and its entries. Any number of cache listeners can be configured. A fully qualified class name may be appended with a JSON specification that will be parsed to become the fields of the parameter to the `init()` method for a class that implements the `Declarable` interface.	

Name	Description	Default Value
--cache-loader	Fully qualified class name of a plug-in to be instantiated for receiving notification of cache misses in the region. At most, one cache loader can be defined in each member for the region. For distributed regions, a cache loader may be invoked remotely from other members that have the region defined. A fully qualified class name may be appended with a JSON specification that will be parsed to become the fields of the parameter to the `initialize()` method for a class that implements the `Declarable` interface.	
--cache-writer	Fully qualified class name of a plug-in to be instantiated for receiving before-event notification of changes to the region and its entries. The plug-in may cancel the event. At most, one cache writer can be defined in each member for the region. A fully qualified class name may be appended with a JSON specification that will be parsed to become the fields of the parameter to the `init()` method for a class that implements the `Declarable` interface.	
--enable-cloning	Determines how `fromDelta` applies deltas to the local cache for delta propagation. When true, the updates are applied to a clone of the value and then the clone is saved to the cache. When false, the value is modified in place in the cache.	`false`
--entry-idle-time-expiration	Number of seconds before a region or an entry expires. Specify `-1` to indicate that there is no expiration of this type	-1
--entry-idle-time-expiration-action	Action that should take place when a region or an entry expires. Select one of the following expiration actions: local-destroy: Removes the region or entry from the local cache, but does not distribute the removal operation to remote members. You cannot use this action on partitioned region entries. destroy: Removes the region or entry completely from the cache. Destroy actions are distributed according to the region's distribution settings. Use this option when the region or entry is no longer needed for any application in the cluster. invalidate: Default expiration action. Marks an entry or all entries in the region as invalid. Distributes the invalidation according to the region's scope. This is the proper choice when the region or the entry is no longer valid for any application in the cluster. local-invalidate: Marks an entry or all entries in the region as invalid but does not distribute the operation. You cannot use this action on partitioned region entries. Local region invalidation is only supported for regions that are not configured as replicated regions.	`invalidate`
--entry-time-to-live-expiration	Number of seconds before a region or an entry expires. Specify `-1` to indicate that there is no expiration of this type.	-1

Name	Description	Default Value
--entry-time-to-live-expiration-action	Action that should take place when a region or an entry expires. Select one of the following expiration actions: local-destroy: Removes the region or entry from the local cache, but does not distribute the removal operation to remote members. You cannot use this action on partitioned region entries. destroy: Removes the region or entry completely from the cache. Destroy actions are distributed according to the region's distribution settings. Use this option when the region or entry is no longer needed for any application in the cluster. invalidate: Default expiration action. Marks an entry or all entries in the region as invalid. Distributes the invalidation according to the region's scope. This is the proper choice when the region or the entry is no longer valid for any application in the cluster. local-invalidate: Marks an entry or all entries in the region as invalid but does not distribute the operation. You cannot use this action on partitioned region entries. Local region invalidation is only supported for regions that are not configured as replicated regions.	`invalidate`
--entry-idle-time-custom-expiry	The name of a class implementing CustomExpiry for entry idle time. Append a JSON string for initialization properties.	
--entry-time-to-live-custom-expiry	The name of a class implementing CustomExpiry for entry time to live. Append a JSON string for initialization properties.	
--eviction-max	Maximum value for the Eviction Attributes that the eviction algorithm uses to determine when to perform its eviction action. The unit of the maximum value is determined by the Eviction Algorithm.	0
--gateway-sender-id	IDs of the Gateway Senders where data is routed.	
--groups	Group(s) of members where the region will be altered.	
--name	Required. Name (including path) of the region.	
--region-idle-time-expiration	Number of seconds before a region or an entry expires. If timeout is not specified, it defaults to zero (which means no expiration).	-1

Name	Description	Default Value
--region-idle-time-expiration-action	Action that should take place when a region or an entry expires. Select one of the following expiration actions: local-destroy: Removes the region or entry from the local cache, but does not distribute the removal operation to remote members. You cannot use this action on partitioned region entries. destroy: Removes the region or entry completely from the cache. Destroy actions are distributed according to the region's distribution settings. Use this option when the region or entry is no longer needed for any application in the cluster. invalidate: Default expiration action. Marks an entry or all entries in the region as invalid. Distributes the invalidation according to the region's scope. This is the proper choice when the region or the entry is no longer valid for any application in the cluster. local-invalidate: Marks an entry or all entries in the region as invalid but does not distribute the operation. You cannot use this action on partitioned region entries. Local region invalidation is only supported for regions that are not configured as replicated regions.	`invalidate`
--region-time-to-live-expiration	Number of seconds before a region or an entry expires. If timeout is not specified, it defaults to zero (which means no expiration).	-1
--region-time-to-live-expiration-action	Action that should take place when a region or an entry expires. Select one of the following expiration actions: local-destroy: Removes the region or entry from the local cache, but does not distribute the removal operation to remote members. You cannot use this action on partitioned region entries. destroy: Removes the region or entry completely from the cache. Destroy actions are distributed according to the region's distribution settings. Use this option when the region or entry is no longer needed for any application in the cluster. invalidate: Default expiration action. Marks an entry or all entries in the region as invalid. Distributes the invalidation according to the region's scope. This is the proper choice when the region or the entry is no longer valid for any application in the cluster. local-invalidate: Marks an entry or all entries in the region as invalid but does not distribute the operation. You cannot use this action on partitioned region entries. Local region invalidation is only supported for regions that are not configured as replicated regions.	`invalidate`

Example Commands:

```
alter region --name=region1 --eviction-max=5000 [-group=all]
```

Sample Output:

```
gfsh>alter region --name=customer --eviction-max=5000
Member  | Status
------- | ----------------------------------
server1 | Region "/customer" altered on "server1"
```

alter runtime

Alters configuration properties for all servers or a subset of servers while the member or members are running. Alter runtime is a cluster configuration command that affects the configuration for newly joining servers. In order for running members to adopt the alteration, they must be stopped and restarted.

The alter runtime command does not apply to locators.

For more information on these configuration properties, see cache.xml and configuration parameter reference.

Availability: Online. You must be connected in `gfsh` to a JMX Manager member to use this command.

Syntax:

```
alter runtime [--members=value(,value)*] [--groups=value(,value)*]
    [--archive-disk-space-limit=value]
    [--archive-file-size-limit=value] [--log-disk-space-limit=value]
    [--log-file-size-limit=value] [--log-level=value]
    [--statistic-archive-file=value] [--statistic-sample-rate=value]
    [--enable-statistics=value] [--copy-on-read(=value)?] [--lock-lease=value]
    [--lock-timeout=value] [--message-sync-interval=value] [--search-timeout=value]
```

Parameters, alter runtime

Name	Description	Default Value
--members	Name or ID of the member(s) whose configuration is to be altered at runtime. If you do not specify this parameter, the configuration properties are modified for all cluster members using the cluster configuration service.	If not specified, all members using the cluster configuration service
--groups	Name of the group(s) whose members's runtime configuration is to be altered. If you do not specify this parameter, the configuration properties are modified for all cluster members using the cluster configuration service.	If not specified, all members using the cluster configuration service
--archive-disk-space-limit	Archive disk space limit. Maximum size (in megabytes) of all inactive statistic archive files combined. If this limit is exceeded, inactive archive files are deleted, oldest first, until the total size is within the limit. If set to zero, disk space use is unlimited. Valid values are (in megabytes): 0 - 1000000.	0
--archive-file-size-limit	Archive file size limit. The maximum size (in megabytes) of a single statistic archive file. Once this limit is exceeded, a new statistic archive file is created, and the current archive file becomes inactive. If set to zero, file size is unlimited. Valid values are (in megabytes): 0 - 1000000.	0

Name	Description	Default Value
--log-disk-space-limit	Log disk space limit. Maximum size in megabytes of all inactive log files combined. If this limit is exceeded, inactive log files are deleted, oldest first, until the total size is within the limit. If set to zero, disk space use is unlimited. Valid values are (in megabytes): 0 - 1000000.	0
--log-file-size-limit	Log file size limit. Maximum size in megabytes of a log file before it is closed and logging rolls on to a new (child) log file. If set to zero, log rolling is disabled. Valid values are (in megabytes): 0 - 1000000.	0
--log-level	The new log level. This option is required and you must specify a value. Valid values are: \`ALL\`, \`TRACE\`, \`DEBUG\`, \`INFO\`, \`WARN\`, \`ERROR\`, \`FATAL\`, \`OFF\`.	INFO
--statistic-archive-file	The file to which the running system member writes statistic samples. For example: "StatisticsArchiveFile.gfs". Must be defined to store the archiving to a file. Adding the `.gz` suffix to the file name causes it to be compressed. See Statistics.	*not set*
--statistic-sample-rate	Statistic sampling rate. Valid values are (in milliseconds): 100 - 60000. See Statistics.	1000
--enable-statistics	Whether statistic sampling should be enabled. Specify `--statistic-archive-file` to store the statistics to a file. Valid values are: `true` and `false`. See Statistics.	false
--copy-on-read	True or false. Sets the copy on read region attribute for cache read operations. See Copy on Read Behavior.	false
--lock-lease	Sets the length, in seconds, of distributed lock leases obtained by this cache. See Setting Cache Timeouts.	120
--lock-timeout	Sets the number of seconds a cache operation may wait to obtain a distributed lock lease before timing out. See Setting Cache Timeouts.	60
--message-sync-interval	Sets the frequency (in seconds) at which a message will be sent by the primary cache-server node to all the secondary cache-server nodes to remove the events which have already been dispatched from the queue. See Change Server Queue Synchronization Frequency.	1
--search-timeout	Sets the number of seconds a cache get operation can spend searching for a value. See Setting Cache Timeouts.	300

Example Commands:

```
alter runtime --members=server1 --log-level=WARN --enable-statistics=true
```

Sample Output:

```
gfsh>alter runtime --members=server1 --log-level=WARN --enable-statistics=true
Runtime configuration altered successfully for the following member(s)
192.0.2.0(server1:240)<v1>:64871
```

backup disk-store

Back up persistent data from all members to the specified directory.

The specified directory must exist on all members, but it can be a local directory on each machine. This command ensures that backup files are not corrupted by concurrent operations. Backing up a running system using the operating system copy command is not recommended.

You can also use this command to perform an incremental backup. See Creating Backups for System Recovery and Operational Management for more information on incremental backup.

Availability: Online. You must be connected in `gfsh` to a JMX Manager member to use this command.

Syntax:

```
backup disk-store --dir=value [--baseline-dir=value]
```

Table 1. Backup Disk-Store Parameters

Name	Description
--dir	*Required.* Directory to which backup files are written.
--baseline-dir	Directory that contains the baseline backup used for comparison during an incremental backup. An incremental backup operation backs up any data that is not present in the directory specified in --baseline-dir. If the member cannot find previously backed up data or if the previously backed up data is corrupt, the command performs a full backup on that member.

Table 1. Backup Disk-Store Parameters

Example Commands:

```
backup disk-store --dir=data/backups

backup disk-store --dir=data/backup/disk-store --baselineDir=data/backups/2012-09-24-1
7-08-50
```

Sample Output:

```
gfsh>backup disk-store --dir=data/backups
The following disk stores were backed up successfully

Member  |                 UUID                 |               Directory
| Host
------- | ------------------------------------ | ------------------------------------
| ---------------
server2 | a6bb11f0-0baa-45c9-b23e-64876d02a586 | c:\PivotalGemFire70\Latest\server2\.
| 192.0.2.0
server1 | 8dc365bd-c086-4af4-99d0-86b0b521aa04 | c:\PivotalGemFire70\Latest\server1\.
| 192.0.2.0
```

change loglevel

Changes the logging level on specified members.

The `change loglevel` command applies only to the members specified, whether they are servers or locators. The change does not apply to unspecified or subsequently-added members.

Availability: Online. You must be connected in `gfsh` to a JMX Manager member to use this command.

Syntax:

```
change loglevel --log-level=value [--members=value(,value)*] [--groups=value(,value)*]
```

Table 1. Change Loglevel Parameters

Name	Description	Default Value
--members	Name or ID of one or more member(s) whose logging level you want to change.	
--groups	One or more group names. The logging level changes for all members of these groups.	
--log-level	*Required.* Log level to change. Valid options are: `ALL`, `TRACE`, `DEBUG`, `INFO`, `WARN`, `ERROR`, `FATAL`, `OFF`.	

Table 1. Change Loglevel Parameters

Example Commands:

```
gfsh>change loglevel --log-level=DEBUG --members=server1
```

Sample Output:

```
gfsh>change loglevel --log-level=DEBUG --members=server1

Summary

              Member              | Changed log-level
--------------------------------- | -----------------
192.0.2.0(server1:3060)<v1>:24653 | true
```

clear defined indexes

Clears all the defined indexes.

Index definitions are stored locally on the `gfsh` client. If you want to create a new set of indexes or if one or more of the index creations fail, you might want to clear the definitions

See also define index.

Availability: Online or offline.

Syntax:

```
clear defined indexes
```

Example Commands:

```
gfsh> clear defined indexes
```

Sample Output:

```
gfsh>clear defined indexes
Index definitions successfully cleared
```

close

Close durable client CQs and durable clients.

- **close durable-client**

  Attempts to close a durable client. The client must be disconnected for this command to work.

- **close durable-cq**

  Closes the durable continuous query (CQ) registered by the durable client and drain events held for the durable CQ from the subscription queue.

close durable-client

Attempts to close a durable client. The client must be disconnected for this command to work.

Availability: Online. You must be connected in `gfsh` to a JMX Manager member to use this command.

Syntax:

```
close durable-client --durable-client-id=value [--members=value(,value)*]
[--groups=value(,value)*]
```

Name	Description
--durable-client-id	*Required.* The ID of the durable client.
--members	Name or ID of the member(s) for which the durable client is to be closed.
--groups	Group(s) of members for which the durable client is to be closed.

Table 1. Close Durable-Client Parameters

Example Commands:

```
close durable-client --durable-client-id=client1
```

Sample Output:

```
gfsh>close durable-client --durable-client-id=client1
Closed the durable client : "client1". on following members.
1.server4
2.server3
```

Error Messages:

```
gfsh>close durable-cq --durable-cq-name=cq1 --durable-client-id=client1

Could not close the durable-cq : "cq1" for the durable-client-id : "client1" due to fo
llowing reasons.

CacheClientProxy: Could not drain cq cq1 because client proxy id client1 is connected.
Occurred on members
1.server4
2.server3

No client found with client-id : client1
Occurred on members
1.server1
```

close durable-cq

Closes the durable continuous query (CQ) registered by the durable client and drain events held for the durable CQ from the subscription queue.

Availability: Online. You must be connected in `gfsh` to a JMX Manager member to use this command.

Syntax:

```
close durable-cq --durable-client-id=value --durable-cq-name=value
[--members=value(,value)*] [--groups=value(,value)*]
```

Name	Description
--durable-client-id	Required. The ID of the durable client.
--durable-cq-name	Required. Name of the CQ to be closed.
--members	Name or ID of the member(s) for which the durable client is registered and the durable CQ to be closed.
--groups	Group(s) of members for which the durable client is registered and the durable CQ to be closed.

Table 2. Close Durable-CQ Parameters

Example Commands:

```
close durable-cq --durable-client-id=client1 --durable-cq-name=cq1
```

Sample Output:

```
gfsh>close durable-cq --durable-cq-name=cq1 --durable-client-id=client1
Closed the durable cq : "cq1" for the durable client : "client1". on following member
s.
1.server4
2.server3
```

Error Messages:

```
gfsh>close durable-client --durable-client-id=client1
Unable to close the durable client : "client1" due to following reasons.

Cannot close a running durable client : client1
Occurred on members
1.server4
2.server3

No client found with client-id : client1
Occurred on members
1.server1
```

compact

Compact online and offline disk-stores.

- **compact disk-store**

  Compact a disk store on all members with that disk store.

- **compact offline-disk-store**

  Compact an offline disk store.

compact disk-store

Compact a disk store on all members with that disk store.

This command uses the compaction threshold that each member has configured for its disk stores. The disk store must have the `allow-force-compaction` property set to `true`.

See Running Compaction on Disk Store Log Files for more information.

Availability: Online. You must be connected in `gfsh` to a JMX Manager member to use this command.

Syntax:

```
compact disk-store --name=value [--groups=value(,value)*]
```

Parameters, compact disk-store

Name	Description
--name	*Required.* Name of the disk store to be compacted.
--groups	Group(s) of members that perform disk compaction. If no group is specified, then the disk store is compacted by all members.

Example Commands:

```
compact disk-store --name=Disk1
compact disk-store --name=Disk1 --group=MemberGroup1,MemberGroup2
```

Error Messages:

```
"Disk store \"{0}\" does not exist.";
 " for group(s) \"{0}\"";
"No members found in the specified group(s) \"{0}\".";
"Compaction was attempted but nothing to compact.";
"Error occurred while doing compaction. Reason: \"{0}\"";
```

compact offline-disk-store

Compact an offline disk store.

If the disk store is large, you may need to allocate additional memory to the process by using the `--J=-XmxNNNm` parameter.

See Running Compaction on Disk Store Log Files for more information.

Note: Do not perform offline compaction on the baseline directory of an incremental backup.

Availability: Online or offline.

Syntax:

```
compact offline-disk-store --name=value --disk-dirs=value(,value)*
[--max-oplog-size=value] [--J=value(,value)*]
```

Parameters, compact offline-disk-store

Name	Description	Default Value
--name	*Required*. Name of the offline disk store to be compacted.	
--disk-dirs	*Required*. One or more directories where data for the disk store was previously written. Separate directories with commas.	
--max-oplog-size	Maximum size (in megabytes) of the oplogs created by compaction.	-1
--J	Arguments passed to the Java Virtual Machine performing the compact operation on the disk store. For example: `--J=-Xmx1024m`.	

Example Commands:

```
compact offline-disk-store --name=Disk2 --disk-dirs=/Disks/Disk2
compact offline-disk-store --name=Disk2 --disk-dirs=/Disks/Disk2 --max-oplog-size=512
--J=-Xmx1024m
```

configure

Configure Portable Data eXchange (PDX) for all the cache(s) in the cluster.

configure pdx

Configures VMware GemFire’s Portable Data eXchange for all the cache(s) in the cluster and persists the pdx configuration in the locator with the cluster configuration service.

For consistent results, PDX should be configured before any servers have started. A server that is running at the time PDX is configured will not adopt the new configuration until it has been restarted.

Availability: Online.

Syntax:

```
configure pdx [--read-serialized=value] [--ignore-unread-fields=value]
    [--disk-store(=value)?] [--auto-serializable-classes=value(,value)*]
    [--portable-auto-serializable-classes=value(,value)*]
```

Parameters, configure pdx:

Name	Description	Default
--read-serialized	When true, PDX deserialization produces a PdxInstance instead of an instance of the domain class.	false
--ignore-unread-fields	Controls whether PDX ignores fields that were unread during deserialization. The default is to preserve unread fields by including their data during serialization. However, if you configure the cache to ignore unread fields, then their data will be lost during serialization. You should set this attribute to true only if you know this member will only be reading cache data. In this use case you do not need to pay the cost of preserving the unread fields, since you will never be reserializing PDX data.	false
--disk-store	Named disk store where the PDX type data will be stored. If specified without a value, then “DEFAULT” is used.	none
--auto-serializable-classes	Configures ReflectionBasedAutoSerializer as the PDX serializer for member classes. Specifies patterns to be matched against domain class names to determine whether they should be auto-serialized. Classes are not checked for portability to non-java languages (equivalent to `check-portability=false`).	none
--portable-auto-serializable-classes	Configures ReflectionBasedAutoSerializer as the PDX serializer for member classes. Specifies patterns to be matched against domain class names to determine whether they should be serialized. Serialization done by the PDX autoserializer will throw an exception if the object of these classes are not portable to non-Java languages (equivalent to `check-portability=true`).	none

Example Commands:

```
gfsh>configure pdx --read-serialized=true
```

Sample Output:

```
gfsh>configure pdx --read-serialized=true
persistent = false
read-serialized = true
ignore-unread-fields = false

gfsh>configure pdx --disk-store=/home/username/server4/DEFAULT.drf
persistent = true
disk-store = /home/username/server4/DEFAULT.drf
read-serialized = false
ignore-unread-fields = false
```

Error Messages:

```
Configure pdx failed because cluster configuration is disabled.
```

```
"Failed to persist the configuration changes due to this command, Revert the command t
o maintain consistency.
Please use "status cluster-config-service" to determing whether Cluster configuration
service is RUNNING."
```

connect

Connect to a JMX manager either directly or via a locator.

If you are connecting via a locator, and a JMX manager does not already exist, the locator starts one.

gfsh connects as a discovery client to the locator service and asks where the JMX Manager is. The locator knows when there is no member currently configured as the JMX manager and simply starts up the JMX manager service within itself. gfsh connects as a JMX client to the locator's JMX RMI port.

You can also connect to a remote locator using the HTTP protocol, as illustrated by the second example below.

Availability: Offline. You will receive a notification "Already connected to: host[port]" if you are already connected.

Syntax:

```
connect [--locator=value] [--jmx-manager=value] [--use-http(=value)?] [--url=value]
    [--user=value][--password=value] [--key-store=value] [--key-store-password=value]
    [--trust-store=value] [--trust-store-password=value] [--ciphers=value]
    [--protocols=value] [--security-properties-file=value] [--use-ssl(=value)?]
    [--skip-ssl-validation(=value)?]
```

Table 1. Connect Parameters

Name	Description	Default
--locator	Network address of the Locator in the form: `host[port]`.	`localhost[10334]`
--jmx-manager	Network address of the JMX manager in the form: `host[port]`.	
--use-http	**Deprecated: inferred by the presence of --url.** Connects to a JMX manager HTTP service using the HTTP protocol.	
--url	URL used to connect to a JMX manager's HTTP service.	
--username --user	The user name of the credential to use in authentication when connecting to the JMX manager. When specified, if the `--password` option is not also specified, `gfsh` will prompt for the password.	

Name	Description	Default
--password	The password portion of the credential to use in authentication when connecting to the JMX manager.	
--key-store	Java keystore file containing this application's certificate and private key. If the `--key-store-password` parameter is not specified, gfsh prompts the operator for the password.	
--key-store-password	Password to access the private key from the keystore file specified by `--key-store`.	
--trust-store	Java keystore file containing the collection of CA certificates trusted by this application. If the `--trust-store-password` parameter is not specified, gfsh prompts the operator for the password.	
--trust-store-password	Password to unlock the keystore file specified by `--trust-store`.	
--ciphers	SSL/TLS ciphers used when encrypting the connection. The default is "any".	
--protocols	SSL/TLS protocol versions to enable when encrypting the connection. The default is "any".	
--security-properties-file	The `gfsecurity.properties` file for configuring gfsh to connect to the Locator/Manager. The file path can be absolute or relative to the current gfsh directory.	
--use-ssl	Whether to use SSL for communication with Locator and/or JMX Manager. If set to `true`, the connect command also reads `gfsecurity.properties`. SSL Options take precedence over values set in the properties file. If none are specified, defaults are used.	• If the parameter is not specified: `false` • If the parameter is specified without a value: `true`
--skip-ssl-validation	When SSL communication is enabled and this option is specified or assigned the value `true`, this gfsh client accepts any SSL certificate, allowing this gfsh client to authenticate any locator or server to which it is connecting. This option exists to facilitate testing, and it is not intended for production systems.	`false`

Table 1. Connect Parameters

Example Commands:

If you do not specify a locator or JMX manager, `gfsh` connects to the locator on the localhost at the default port.

```
gfsh>connect
```

Sample Output:

```
gfsh>connect
Connecting to Locator at [host=localhost, port=10334] ..
Connecting to Manager at [host=GemFireStymon, port=1099] ..
Successfully connected to: [host=GemFireStymon, port=1099]
```

Example of connecting to a remote locator over HTTP:

```
gfsh>connect --use-http=true --url="http://myLocatorHost.example.com:8080/gemfire/v1"
Successfully connected to: GemFire Manager's HTTP service @
http://myLocatorHost.example.com:8080/gemfire/v1
```

Error Messages:

```
"Locator could not find a JMX Manager";
"jmx password must be specified.";
"Could not connect to : {0}. {1}";
"Could not find a GemFire jmx-manager service running at {0}.";
"Could not connect to GemFire Locator service at {0}."
```

create

Create async-event-queues, disk-stores, gateway receivers, gateway senders, indexes, and regions.

- **create async-event-queue**

  Creates an asynchronous event queue for batching events before they are delivered by a gateway sender.

- **create defined indexes**

  Creates all the defined indexes.

- **create disk-store**

  Defines a pool of one or more disk stores, which can be used by regions and client subscription queues, and gateway sender queues for WAN distribution.

- **create gateway-receiver**

  Creates a gateway receiver. You can only have one gateway receiver on each member, and unlike a gateway sender, you do not need to specify an identifier for the gateway receiver .

- **create gateway-sender**

  Creates a gateway sender on one or more members of a cluster.

- **create index**

Create an index that can be used when executing queries.

- **create jndi-binding**

  Create a JNDI binding that specifies resource attributes which describe a JDBC connection.

- **create lucene index**

  Create a region with given path and configuration.

- **create region**

  Create a region with given path and configuration.

Note: The order in which components are created matters. For example, the recommendation for WAN setup is:

- Create/start WAN senders first
- Create Regions
- Create/start WAN receivers last

This assures that when WAN receivers are started, their associated regions are in place. Otherwise, the `create region` command may fail if events are received before the region exists. For more on this topic, see Configuring a Multi-site (WAN) System.

create async-event-queue

Creates an asynchronous event queue for batching events before they are delivered by a gateway sender.

See Configuring Multi-Site (WAN) Event Queues.

Availability: Online. You must be connected in `gfsh` to a JMX Manager member to use this command.

Syntax:

```
create async-event-queue --id=value --listener=value [--groups=value(,value)*]
    [--parallel(=value)?] [--enable-batch-conflation(=value)?] [--batch-size=value]
    [--batch-time-interval=value] [--persistent(=value)?] [--disk-store=value]
    [--disk-synchronous(=value)?] [--max-queue-memory=value]
    [--dispatcher-threads=value] [--order-policy=value]
    [--gateway-event-filter=value(,value)*]
    [--gateway-event-substitution-filter=value]
    [--listener-param=value(,value)*] [--forward-expiration-destroy(=value)?]
    [--pause-event-processing(=value)?]
```

Parameters, create async-event-queue:

Name	Description	Default Value
--id	*Required*. ID of the asynchronous event queue	
--groups	The queue is created on all members of the group(s). If you do not specify a group, the queue is created on all members.	

Name	Description	Default Value
--parallel	Specifies whether the queue is parallel.	false
--enable-batch-conflation	Enables batch conflation.	false
--batch-size	Maximum number of messages that a batch can contain.	100
--batch-time-interval	Maximum amount of time, in ms, that can elapse before a batch is delivered.	5
--persistent	Boolean value that determines whether VMware GemFire persists this queue.	false If specified with out a value, default is true.
--disk-store	Named disk store to use for storing queue overflow, or for persisting the queue. If you specify a value, the named disk store must exist. If you specify a null value, VMware GemFire uses the default disk store for overflow and queue persistence.	
--disk-synchronous	Specifies whether disk writes are synchronous.	true
--max-queue-memory	Maximum amount of memory in megabytes that the queue can consume before overflowing to disk.	100
--dispatcher-threads	Number of threads used for sending events.	5
--order-policy	Policy for dispatching events when `--dispatcher-threads` is > 1. Possible values are `THREAD`, `KEY`, `PARTITION`.	KEY
--gateway-event-filter	List of fully qualified class names of GatewayEventFilters for this queue. These classes filter events before dispatching to remote servers.	
--gateway-event-substitution-filter	Fully-qualified class name of the `GatewayEventSubstitutionFilter` for this queue.	
--listener	*Required.* Fully-qualified class name of Async Event Listener for this queue	
--listener-param	Parameter name and value to be passed to the Async Event Listener class. Optionally, you can specify a value by following the parameter name with the `#` character and the value. For example: `--listener-param=myParam#24`	

Name	Description	Default Value
--forward-expiration-destroy	Enables forwarding of expiration destroy operations to AsyncEventListener instances. If specified without a value, this parameter is set to “false”.	false
--pause-event-processing	Specifies whether event dispatching from the queue to the listener(s) will be paused when the AsyncEventQueue is started. If specified without a value, this parameter is set to "true".	false

Example Commands:

```
create async-event-queue --id=myAEQ --listener=myApp.myListener
```

create defined indexes

Creates all the defined indexes.

See also define index and clear defined indexes.

Availability: Online. You must be connected in gfsh to a JMX Manager member to use this command.

Syntax:

```
create defined indexes [--members=value(,value)*] [--groups=value(,value)*]
```

Parameters, create defined indexes:

Name	Description	Default
--members	Name/Id of the member(s) on which index will be created.	
--groups	The index will be created on all the members in the member group(s).	

Example Commands:

```
create defined indexes
```

Sample Output:

```
gfsh>create defined indexes
Indexes successfully created. Use list indexes to get details.
1. ubuntu(server1:17682)<v1>:27574
```

If index creation fails, you may receive an error message in gfsh similar to the following:

```
gfsh>create defined indexes
Exception : org.apache.geode.cache.query.RegionNotFoundException ,
Message : Region ' /r3' not found: from  /r3Occurred on following members
1. india(s1:17866)<v1>:27809
```

create disk-store

Defines a pool of one or more disk stores, which can be used by regions and client subscription queues, and gateway sender queues for WAN distribution.

See Disk Storage

Availability: Online. You must be connected in gfsh to a JMX Manager member to use this command.

Syntax:

```
create disk-store --name=value --dir=value(,value)* [--allow-force-compaction(=valu
e)?]
[--auto-compact(=value)?] [--compaction-threshold=value] [--max-oplog-size=value]
[--queue-size=value] [--time-interval=value] [--write-buffer-size=value]
[--groups=value(,value)*]
[--disk-usage-warning-percentage=value] [--disk-usage-critical-percentage=value]
```

Parameters, create disk-store:

Name	Description	Default Value
--name	*Required.* The name of this disk store.	
--dir	*Required.* One or more directory names where the disk store files are written. Optionally, directory names may be followed by `#` and the maximum number of megabytes that the disk store can use in the directory. For example: `--dir=/data/ds1` `--dir=/data/ds2#5000` If the specified directory does not exist, the command will create the directory for you.	If the maximum directory size in megabytes is not specified, it will be set to `2147483647` (the value of `Integer.MAX_VALUE`)
--allow-force-compaction	Set to true to allow disk compaction to be forced on this disk store.	false
--auto-compact	Set to true to automatically compact the disk files.	true
--compaction-threshold	Percentage (0..100) of live data (non-garbage content) remaining in the operation log, below which it is eligible for compaction. As garbage is created (by entry destroys, entry updates, and region destroys and creates) the percentage of remaining live data declines. Falling below this percentage initiates compaction if auto-compaction is turned on. If not, the file will be eligible for manual compaction at a later time.	50
--max-oplog-size	Maximum size, in megabytes, for an oplog file. When the oplog file reaches this size, the file is rolled over to a new file.	1024
--queue-size	Maximum number of operations that can be asynchronously queued to be written to disk.	0
--time-interval	The number of milliseconds that can elapse before unwritten data is written to disk.	1000

Name	Description	Default Value
--groups	The disk store is created on all members of the group(s). If no group is specified, the disk store is created on all members.	
--write-buffer-size	The size in bytes of the write buffer that this disk store uses when writing data to disk. Larger values may increase performance but use more memory. The disk store allocates one direct memory buffer of this size.	32768
--disk-usage-warning-percentage	Disk usage above this threshold generates a warning message. For example, if the threshold is set to 90%, then on a 1 TB drive falling under 100 GB of free disk space generates the warning. Set to "0" (zero) to disable.	90
--disk-usage-critical-percentage	Disk usage above this threshold generates an error message and shuts down the member's cache. For example, if the threshold is set to 99%, then falling under 10 GB of free disk space on a 1 TB drive generates the error and shuts down the cache. Set to "0" (zero) to disable.	99

Example Commands:

```
create disk-store --name-store1 --dir=/data/ds1
```

Sample Output:

```
gfsh>create disk-store --name-store1 --dir=/data/ds1
Member  | Result
------- | -------
server1 | Success
```

create gateway-receiver

Creates gateway receivers. You can only have one gateway receiver on each member, and unlike a gateway sender, you do not need to specify an identifier for the gateway receiver.

The create occurs on all servers, unless the `--groups` or `--members` option is specified.

If the gateway receiver creation succeeds on at least one member, this `gfsh` command exits with an exit code indicating success.

Outputs a tabular format status of each member’s gateway receiver, independent of the success or failure of the creation.

See Gateway Receivers.

Availability: Online. You must be connected in `gfsh` to a JMX Manager member to use this command.

Syntax:

```
create gateway-receiver [--groups=value(,value)*] [--members=value(,value)*]
  [--manual-start=(value)?] [--start-port=value] [--end-port=value] [--bind-address=va
lue]
  [--maximum-time-between-pings=value] [--socket-buffer-size=value]
  [--gateway-transport-filter=value(,value)*] [--hostname-for-senders=value]
  [--if-not-exists=(value)?]
```

Parameters, create gateway-receiver:

Name	Description	Default Value
--groups	Gateway receivers are created on the members of the group(s).	
--members	Name of the member(s) on which to create the gateway receiver. For backward compatibility, no gateway receiver configuration is persisted if this option is specified and cluster configuration is enabled.	
--manual-start	Boolean value that specifies whether you need to manually start the gateway receiver. When specified without providing a boolean value or when specified and set to "true", the gateway receiver must be started manually.	false
--start-port	Starting port number to use when specifying the range of possible port numbers this gateway receiver will use to connects to gateway senders in other sites. VMware GemFire chooses an unused port number in the specified port number range to start the receiver. If no port numbers in the range are available, an exception is thrown. The `start-port` and `end-port` values are inclusive. For example, if you specify `start-port="50510"` and `end-port="50520"`, VMware GemFire chooses a port value from 50510 to 50520.	5000
--end-port	Defines the upper bound port number to use when specifying the range of possible port numbers this gateway receiver will use to for connections from gateway senders in other sites. VMware GemFire chooses an unused port number in the specified port number range to start the receiver. If no port numbers in the range are available, an exception is thrown. The `end-port` and `start-port` values are inclusive. For example, if you specify `start-port="50510"` and `end-port="50520"`, VMware GemFire chooses a port value from 50510 to 50520.	5500
--bind-address	Network address for connections from gateway senders in other sites. Specify the address as a literal string value.	
--socket-buffer-size	An integer value that sets the buffer size (in bytes) of the socket connection for this gateway receiver. This value should match the `socket-buffer-size` setting of gateway senders that connect to this receiver.	32768
--gateway-transport-filter	The fully qualified class name of the GatewayTransportFilter to be added to the Gateway receiver.	
--maximum-time-between-pings	Integer value that specifies the time interval (in milliseconds) to use between pings to connected WAN sites. This value determines the maximum amount of time that can elapse before a remote WAN site is considered offline.	60000
--hostname-for-senders	The host name or IP address told to gateway senders as the address for them to connect to. The locator informs gateway senders of this value.	

Name	Description	Default Value
--if-not-exists	When specified without providing a boolean value or when specified and set to "true", gateway receivers will not be created if they already exist. Command output reports the status of each creation attempt.	false

Example Commands:

```
gfsh>create gateway-receiver --members=server1
```

Sample Output:

```
gfsh>create gateway-receiver --members=server1
Member  | Status
------- | ------------------------------------------------------------------------
server1 | GatewayReceiver created on member "server1" and will listen on the port "0"
```

create gateway-sender

Creates a gateway sender on one or more members of a cluster.

See Gateway Senders.

Note: The gateway sender configuration for a specific sender `id` must be identical on each VMware GemFire member that hosts the gateway sender.

Availability: Online. You must be connected in `gfsh` to a JMX Manager member to use this command.

Syntax:

```
create gateway-sender --id=value --remote-distributed-system-id=value
   [--groups=value(,value)*] [--members=value(,value)*] [--parallel(=value)?]
   [--manual-start=value] [--socket-buffer-size=value] [--socket-read-timeout=value]
   [--enable-batch-conflation=value] [--batch-size=value] [--batch-time-interval=valu
e]
   [--enable-persistence=value] [--disk-store-name=value] [--disk-synchronous=value]
   [--maximum-queue-memory=value] [--alert-threshold=value] [--dispatcher-threads=valu
e]
   [--order-policy=value][--gateway-event-filter=value(,value)*]
   [--gateway-transport-filter=value(,value)*]
   [--group-transaction-events(=value)?]
```

Parameters, create gateway-sender:

Name	Description	Default
--id	*Required.* Unique identifier for the gateway sender, usually an identifier associated with a physical location.	
--remote-distributed-system-id	*Required.* ID of the remote cluster where this gateway sender sends events.	
--groups	Gateway senders are created on the members of the group(s).	

Name	Description	Default
--members	Name of the member(s) on which to create the gateway sender.	
--parallel	When set to true, specifies a parallel Gateway Sender.	false
--enable-batch-conflation	Boolean value that determines whether VMware GemFire should conflate messages.	false
--manual-start	**Deprecated.** Boolean value that specifies whether you need to manually start the gateway sender. If you supply a null value, the default value of false is used, and the gateway sender starts automatically. *A manual start is likely to cause data loss, so manual start should never be used in a production system.*	false
--socket-buffer-size	Size of the socket buffer that sends messages to remote sites. This size should match the size of the `socket-buffer-size` attribute of remote gateway receivers that process region events.	32768
--socket-read-timeout	Amount of time in milliseconds that the gateway sender will wait to receive an acknowledgment from a remote site. By default this is set to 0, which means there is no timeout. If you do set this timeout, you must set it to a minimum of 30000 (milliseconds). Setting it to a lower number will generate an error message and reset the value to the default of 0.	0
--batch-size	Maximum number of messages that a batch can contain.	100
--batch-time-interval	Maximum number of milliseconds that can elapse between sending batches.	1000
--enable-persistence	Boolean value that determines whether VMware GemFire persists the gateway queue.	false
--disk-store-name	Named disk store to use for storing the queue overflow, or for persisting the queue. If you specify a value, the named disk store must exist. If you specify a null value, VMware GemFire uses the default disk store for overflow and queue persistence.	
--disk-synchronous	For regions that write to disk, boolean that specifies whether disk writes are done synchronously for the region.	true
--maximum-queue-memory	Maximum amount of memory in megabytes that the queue can consume before overflowing to disk.	100 MB
--alert-threshold	Maximum number of milliseconds that a region event can remain in the gateway sender queue before VMware GemFire logs an alert.	0
--dispatcher-threads	Number of dispatcher threads that are used to process region events from a gateway sender queue or asynchronous event queue.	5

Name	Description	Default
--order-policy	When the `dispatcher-threads` attribute is greater than 1, `order-policy` configures the way in which multiple dispatcher threads process region events from a serial gateway queue or serial asynchronous event queue. This attribute can have one of the following values: **key** When distributing region events from the local queue, multiple dispatcher threads preserve the order of key updates. **thread** When distributing region events from the local queue, multiple dispatcher threads preserve the order in which a given thread added region events to the queue. **partition** When distributing region events from the local queue, multiple dispatcher threads preserve the order in which region events were added to the local queue. For a partitioned region, this means that all region events delivered to a specific partition are delivered in the same order to the remote VMware GemFire site. For a distributed region, this means that all key updates delivered to the local gateway sender queue are distributed to the remote site in the same order. You cannot configure the `order-policy` for a parallel event queue, because parallel queues cannot preserve event ordering for regions. Only the ordering of events for a given partition (or in a given queue of a distributed region) can be preserved.	key
--gateway-event-filter	A list of fully-qualified class names of GatewayEventFilters (separated by commas) to be associated with the GatewaySender. This serves as a callback for users to filter out events before dispatching to a remote cluster. For example: `gateway-event-filter=com.user.filters.MyFilter1,com.user.filters.MyFilters2`	
--gateway-transport-filter	The fully-qualified class name of the GatewayTransportFilter to be added to the GatewaySender.	

Name	Description	Default
--group-transaction-events	Boolean value to ensure that all the events of a transaction are sent in the same batch, i.e., they are never spread across different batches. Only allowed to be set on gateway senders with the `parallel` attribute set to false and `dispatcher-threads` attribute equal to 1, or on gateway senders with the `parallel` attribute set to true. Also, the `enable-batch-conflation` attribute of the gateway sender must be set to false. **Note:** In order to work for a transaction, the regions to which the transaction events belong must be replicated by the same set of senders with this flag enabled. **Note:** If the above condition is not fulfilled or under very high load traffic conditions, it may not be guaranteed that all the events for a transaction will be sent in the same batch, even if `group-transaction-events` is enabled. The number of batches sent with incomplete transactions can be retrieved from the `GatewaySenderMXBean` bean.	false
--enforce-threads-connect-same-receiver	This parameter applies only to serial gateway senders. If true, receiver member id is checked by all dispatcher threads when the connection is established to ensure they connect to the same receiver. Instead of starting all dispatcher threads in parallel, one thread is started first, and after that the rest are started in parallel.	false

Example Commands:

```
gfsh>create gateway-sender --remote-distributed-system-id="2" --id="sender2"
```

Sample Output:

```
gfsh>create gateway-sender --remote-distributed-system-id="2" --id="sender2"
Member  | Status
------- | --------------------------------------------
server1 | GatewaySender "sender2" created on "server1"
```

create index

Create an index that can be used when executing queries.

Availability: Online. You must be connected in `gfsh` to a JMX Manager member to use this command.

See Working with Indexes.

Syntax:

```
create index --name=value --expression=value --region=value
[--members=value(,value)*] [--type=value] [--groups=value(,value)*]
```

Parameters, create index:

Name	Description	Default
--name	*Required.* Name of the index to create.	
--expression	*Required.* Field of the region values that are referenced by the index.	
--region	*Required.* Name/Path of the region which corresponds to the "from" clause in a query.	
--members	Name/Id of the member(s) on which index will be created.	
--type	Type of the index. Valid values are: `range` and `key`. (A third type, `hash`, is still recognized but hash indexes are deprecated.)	`range`
--groups	The index will be created on all the members in the group(s).	

Example Commands:

```
create index --name=myKeyIndex --expression=region1.Id --region=region1 --type=key
```

Sample Output:

```
gfsh>create index --name=myKeyIdex --expression=region1.Id --region=region1 --type=key
Index successfully created with following details
Name       : myKeyIdex
Expression : region1.Id
RegionPath : /region1
Members which contain the index
1. ubuntu(server1:17682)<v1>:27574

gfsh>create index --name=myIndex2 --expression=exp2 --region=/exampleRegion
Failed to create index "myIndex2" due to following reasons
Index "myIndex2" already exists.  Create failed due to duplicate name.
Occurred on following members
1. ubuntu(server1:17682)<v1>:27574
```

create jndi-binding

Create a JNDI binding that specifies resource attributes which describe a JDBC connection.

Availability: Online. You must be connected in gfsh to a JMX Manager member to use this command.

Syntax:

```
create jndi-binding --name=value --url=value
 [--jdbc-driver-class=value] [--type=value] [--blocking-timeout-seconds=value]
 [--conn-pooled-datasource-class=value] [--idle-timeout-seconds=value]
 [--init-pool-size=value] [--login-timeout-seconds=value]
 [--managed-conn-factory-class=value] [--max-pool-size=value] [--password=value]
 [--transaction-type=value] [--username=value] [--xa-datasource-class=value]
 [--if-not-exists(=value)?] [--datasource-config-properties=value(,value)*]
```

Parameters, create jndi-binding:

Name	Description	Default
--name	*Required.* Name of the binding to create.	
--url or --connection-url	*Required.* the JDBC driver connection URL string. For example, `jdbc:hsqldb:hsql://localhost:1701`.	
--jdbc-driver-class	The fully qualified name of the JDBC driver class.	
--type	Type of the XA datasource. One of: `MANAGED`, `SIMPLE`, `POOLED`, or `XAPOOLED`. If `--type=POOLED` and a `--conn-pooled-datasource-class` option is not specified, a pool will be created using Hikari. For more information on Hikari, see https://github.com/brettwooldridge/HikariCP.	`SIMPLE`
--blocking-timeout-seconds	Specifies the maximum time, in seconds, to block while waiting for a connection before throwing an exception.	
--conn-pooled-datasource-class	The fully qualified name of the connection pool implementation that holds XA datasource connections. If `--type=POOLED`, then this class must implement `org.apache.geode.datasource.PooledDataSourceFactory`.	
--idle-timeout-seconds	Specifies the time, in seconds, that a connection may be idle before being closed.	
--init-pool-size	Specifies the initial number of connections the pool should hold.	
--login-timeout-seconds	The quantity of seconds after which the client thread will be disconnected due to inactivity.	
--managed-conn-factory-class	The fully qualified name of the connection factory implementation.	
--max-pool-size	The maximum number of connections that may be created in a pool.	
--password	The default password used when creating a new connection.	
--transaction-type	Type of the transaction. One of `XATransaction`, `NoTransaction`, or `LocalTransaction`.	
--username	Specifies the user name to be used when creating a new connection. When specified, if the `--password` option is not also specified, gfsh will prompt for the password.	
--xa-datasource-class	The fully qualified name of the `javax.sql.XADataSource` implementation class.	
--if-not-exists	When true, a duplicate jndi binding will not be created if one with the same name already exists. When false, an attempt to create a duplicate jndi binding results in an error. The option is set to true if the option is specified without a value.	false

Name	Description	Default
--datasource-config-properties	Properties for the custom `XADataSource` driver. Append a JSON string containing a (name, type, value) tuple to set any property. If `--type=POOLED`, the properties will configure the database data source. If `--type=POOLED` and the value of a name within the tuple begins with the string "pool.", then the properties will configure the pool data source. For example: `--datasource-config-properties={'name':'name1','type':'type1','value':'value1'},{'name':'pool.name2','type':'type2','value':'value2'}`	

Example Commands:

```
gfsh>create jndi-binding --name=jndi1 --type=SIMPLE \
  --jdbc-driver-class=org.apache.derby.jdbc.EmbeddedDriver \
  --url="jdbc:derby:newDB;create=true"
```

create lucene index

Create a Lucene index. For details on Lucene index creation, see Apache Lucene Integration.

For additional Lucene-related gfsh commands, see describe lucene index, destroy lucene index, list lucene indexes and search lucene.

Availability: Online. You must be connected in gfsh to a JMX Manager member to use this command.

Syntax:

```
create lucene index --name=value --region=value --field=value(,value)*
  [--analyzer=value(,value)*] [--serializer=value] [--group=value(,value)*]
```

Parameters, create lucene index:

Name	Description	Default
--name	*Required.* Name of the index to create.	
--region	*Required.* Name/Path of the region on which to define the index.	
--field	*Required.* Field(s) of the region values that are referenced by the index, specified as a comma-separated list. To treat the entire value as a single field, specify `__REGION_VALUE_FIELD`.	
--analyzer	Analyzer(s) to extract terms from text, specified as a comma-separated list. If not specified, the default analyzer is used for all fields. If specified, the number of analyzers must exactly match the number of fields specified. When listing analyzers, use the keyword `DEFAULT` for any field that will use the default analyzer.	Lucene `StandardAnalyzer`
--serializer	Fully qualified class name of the serializer to be used with this index. The serializer must implement the `LuceneSerializer` interface. You can use the built-in `org.apache.geode.cache.lucene.FlatFormatSerializer` to index and search collections and nested fields. If not specified, the simple default serializer is used, which indexes and searches only the top level fields of the region objects.	simple serializer
--group	The index will be created on all the members in the specified member groups.	

Example Commands:

```
gfsh>create lucene index --name=customerIndex --region=/Customer
   --field=__REGION_VALUE_FIELD

gfsh>create lucene index --name=analyzerIndex --region=/Person
     --field=name,email,address,revenue
     --analyzer=DEFAULT,org.apache.lucene.analysis.core.KeywordAnalyzer,
                examples.MyCharacterAnalyzer,DEFAULT
```

Sample Output:

```
gfsh>create lucene index --name=testIndex --region=testRegion
    --field=__REGION_VALUE_FIELD
               Member                  | Status
-------------------------------------- | ---------------------------------
192.168.1.23(server505:17200)<v1>:1025 | Successfully created lucene index
```

create region

Create a region with given path and configuration.

You must specify either a `--type` or a `--template-region` for initial configuration when creating a region. Specifying a `--key-constraint` and `--value-constraint` makes object type information available during querying and indexing.

See Region Data Storage and Distribution.

See Specifying JSON within Command-Line Options for syntax details.

Availability: Online. You must be connected in `gfsh` to a JMX Manager member to use this command.

Syntax:

```
create region --name=value [--type=value] [--template-region=value]
   [--groups=value(,value)*] [--if-not-exists(=value)?]
   [--key-constraint=value] [--value-constraint=value]
   [--enable-statistics=value] [--entry-idle-time-expiration=value]
   [--entry-idle-time-expiration-action=value]
   [--entry-time-to-live-expiration=value]
   [--entry-time-to-live-expiration-action=value]
   [--entry-idle-time-custom-expiry=value] [--entry-time-to-live-custom-expiry=value]
   [--region-idle-time-expiration=value]
   [--region-idle-time-expiration-action=value]
   [--region-time-to-live-expiration=value]
   [--region-time-to-live-expiration-action=value] [--disk-store=value]
   [--enable-synchronous-disk=value] [--enable-async-conflation=value]
   [--enable-subscription-conflation=value] [--cache-listener=value(,value)*]
   [--cache-loader=value] [--cache-writer=value]
   [--async-event-queue-id=value(,value)*]
   [--gateway-sender-id=value(,value)*] [--enable-concurrency-checks=value]
   [--enable-cloning=value] [--concurrency-level=value]
   [--colocated-with=value] [--local-max-memory=value]
   [--recovery-delay=value] [--redundant-copies=value]
   [--startup-recovery-delay=value] [--total-max-memory=value]
   [--total-num-buckets=value] [--compressor=value] [--off-heap(=value)?]
   [--partition-listener=value(,value)*] [--partition-resolver=value]
   [--eviction-entry-count=value] [--scope=value]
```

```
    [--eviction-max-memory=value] [--eviction-action=value]
    [--eviction-object-sizer=value]
```

Parameters, create region:

Name	Description	Default
--name	*Required.* Name/Path of the region to be created.	
--type	*Required* (if template-region is not specified.) Type of region to create. Options include: PARTITION, PARTITION_REDUNDANT, REPLICATE, LOCAL, etc. To get a list of of all region type options, add the --type parameter and then select the TAB key to display a full list.	
--template-region	*Required* (if type is not specified.) Name/Path of the region whose attributes should be duplicated when creating this region.	
--groups	Group(s) of members on which the region will be created.	
--if-not-exists	A new region will not be created if a region with the same name already exists. By default, an attempt to create a duplicate region is reported as an error. If this option is specified without a value or is specified with a value of `true`, then gfsh displays a "Skipping..." acknowledgement, but does not throw an error.	false
--key-constraint	Fully qualified class name of the objects allowed as region keys. Ensures that keys for region entries are all of the same class.	
--value-constraint	Fully qualified class name of the objects allowed as region values. If not specified, then region values can be of any class.	
--enable-statistics	Whether to gather statistics for the region. Must be true to use expiration on the region.	
--entry-idle-time-expiration	How long, in seconds, the region's entries can remain in the cache without being accessed.	no expiration
--entry-idle-time-expiration-action	Action to be taken on an entry that has exceeded the idle expiration. Valid expiration actions include destroy, local-destroy, invalidate (default), local-invalidate.	
--entry-time-to-live-expiration	How long, in seconds, the region's entries can remain in the cache without being accessed or updated. The default is no expiration of this type.	no expiration
--entry-time-to-live-expiration-action	Action to be taken on an entry that has exceeded the TTL expiration. Valid expiration actions include destroy, local-destroy, invalidate (default), local-invalidate.	
--entry-idle-time-custom-expiry	The name of a class implementing CustomExpiry for entry idle time. Append a JSON string for initialization properties.	

Name	Description	Default
--entry-time-to-live-custom-expiry	The name of a class implementing CustomExpiry for entry time to live. Append a JSON string for initialization properties.	
--region-idle-time-expiration	How long, in seconds, the region can remain in the cache without its entries being accessed. The default is no expiration of this type.	
--region-idle-time-expiration-action	Action to be taken on a region that has exceeded the idle expiration. Valid expiration actions include destroy, local-destroy, invalidate (default), local-invalidate. The destroy and local-destroy actions destroy the region. The invalidate and local-invalidate actions leave the region in place, but invalidate all of its entries.	
--region-time-to-live-expiration	How long, in seconds, the region can remain in the cache without its entries being accessed or updated. The default is no expiration of this type.	no expiration
--region-time-to-live-expiration-action	Action to be taken on a region that has exceeded the TTL expiration. Valid expiration actions include destroy, local-destroy, invalidate (default), local-invalidate. The destroy and local-destroy actions destroy the region. The invalidate and local-invalidate actions leave the region in place, but invalidate all of its entries.	
--disk-store	Disk Store to be used by this region. The list disk-stores command can be used to display existing disk stores.	
--enable-synchronous-disk	Whether writes are done synchronously for regions that persist data to disk.	
--enable-async-conflation	Whether to allow aggregation of asynchronous TCP/IP messages sent by the producer member of the region. A false value causes all asynchronous messages to be sent individually.	
--enable-subscription-conflation	Whether the server should conflate its messages to the client. A false value causes all server-client messages to be sent individually.	
--cache-listener	Fully qualified class name of a plug-in to be instantiated for receiving after-event notification of changes to the region and its entries. Any number of cache listeners can be configured. A fully qualified class name may be appended with a JSON specification that will be parsed to become the fields of the parameter to the `init()` method for a class that implements the `Declarable` interface.	

Name	Description	Default
--cache-loader	Fully qualified class name of a plug-in to be instantiated for receiving notification of cache misses in the region. At most, one cache loader can be defined in each member for the region. For distributed regions, a cache loader may be invoked remotely from other members that have the region defined. A fully qualified class name may be appended with a JSON specification that will be parsed to become the fields of the parameter to the `initialize()` method for a class that implements the `Declarable` interface.	
--cache-writer	Fully qualified class name of a plug-in to be instantiated for receiving before-event notification of changes to the region and its entries. The plug-in may cancel the event. At most, one cache writer can be defined in each member for the region. A fully qualified class name may be appended with a JSON specification that will be parsed to become the fields of the parameter to the `init()` method for a class that implements the `Declarable` interface.	
--async-event-queue-id	IDs of the Async Event Queues that will be used for write-behind operations.	
--gateway-sender-id	IDs of the Gateway Senders to which data will be routed.	
--enable-concurrency-checks	Whether Region Version Vectors are implemented. Region Version Vectors are an extension to the versioning scheme that aid in synchronization of replicated regions.	
--enable-cloning	Determines how fromDelta applies deltas to the local cache for delta propagation. When true, the updates are applied to a clone of the value and then the clone is saved to the cache. When false, the value is modified in place in the cache.	
--concurrency-level	Estimate of the maximum number of application threads that will concurrently access a region entry at one time. This attribute does not apply to partitioned regions.	
--colocated-with	Central Region with which this region should be colocated.	
--local-max-memory	Maximum amount of memory, in megabytes, to be used by the region in this process. (The default is 90% of available heap.)	
--recovery-delay	Delay in milliseconds that existing members will wait after a member crashes before restoring this region's redundancy on the remaining members. The default value (-1) indicates that redundancy will not be recovered after a failure.	
--redundant-copies	Number of extra copies of buckets desired. Extra copies allow for both high availability in the face of VM departure (intended or unintended) and load balancing read operations. (Allowed values: 0, 1, 2 and 3)	

Name	Description	Default
--startup-recovery-delay	Delay in milliseconds that new members will wait before assuming their share of cluster-level redundancy. This allows time for multiple regions to start before the redundancy workload is parceled out to the new members. A value of -1 indicates that adding new members will not trigger redundancy recovery.	The default is to recover redundancy immediately when a new member is added.
--total-max-memory	Maximum amount of memory, in megabytes, to be used by the region in all processes.	
--total-num-buckets	Total number of hash buckets to be used by the region in all processes.	113
--compressor	Java class name that implements compression for the region. You can write a custom compressor that implements `org.apache.geode.compression.Compressor` or you can specify the Snappy compressor (`org.apache.geode.compression.SnappyCompressor`), which is bundled with VMware GemFire. See Region Compression.	no compression
--off-heap	Specifies whether the region values are stored in heap memory or off-heap memory. When true, region values are in off-heap memory. If the parameter is specified without a value, the value of true is used.	false
--partition-listener	Specifies fully-qualified class names of one or more custom partition listeners.	
--partition-resolver	Specifies the full path to a custom partition resolver. Specify `org.apache.geode.cache.util.StringPrefixPartitionResolver` to use the included string prefix `PartitionResolver`.	
--eviction-entry-count	Enables eviction, where the eviction policy is based on the number of entries in the region.	
--eviction-max-memory	Enables eviction, where the eviction policy is based on the amount of memory consumed by the region, specified in megabytes.	
--eviction-action	Action to take when the eviction threshold is reached.	
	local-destroy Entry is destroyed locally. Use with caution - may lead to inconsistencies.	
	overflow-to-disk Entry is overflowed to disk. For partitioned regions, this provides the most reliable read behavior across the region.	
--eviction-object-sizer	Specifies your implementation of the ObjectSizer interface to measure the size of objects in the region. The sizer applies only to heap and memory based eviction.	

Name	Description	Default
--scope	Specifies the scope of the Replicated region. This field can be only used if the --type parameter is set with a replicated region type. This parameter is invalid for all other region types. If this parameter is not set and --type is set to a replicated region, the the default scope DISTRIBUTED_ACK is set.	

Example Commands:

```
create region --name=region1 --type=REPLICATE_PERSISTENT \
--cache-writer=org.apache.geode.examples.MyCacheWriter \
--group=Group1 --disk-store=DiskStore1

create region --name=region12 --template-region=/region1

create region --name=region2 --type=REPLICATE \
--cache-listener=org.apache.geode.examples.MyCacheListener1,\
org.apache.geode.examples.MyCacheListener2 \
--group=Group1,Group2

create region --name=region3 --type=PARTITION_PERSISTENT --redundant-copies=2 \
--total-max-memory=1000 --startup-recovery-delay=5 --total-num-buckets=100 \
--disk-store=DiskStore2 --cache-listener=org.apache.geode.examples.MyCacheListener3 \
--group=Group2

create region --name=region4 --type=REPLICATE_PROXY \
--cache-listener=org.apache.geode.examples.MyCacheListener1 --group=Group1,Group2

create region --name=myRegion --type=REPLICATE --eviction-max-memory=100 \
--eviction-action=overflow-to-disk --eviction-object-sizer=my.company.geode.MySizer

create region --name=r1 --type=PARTITION \
--cache-loader=org.example.myLoader{'URL':'jdbc:cloudscape:rmi:MyData'}
```

Sample Output:

```
gfsh>create region --name=myRegion --type=LOCAL
Member  | Status
------- | --------------------------------------
server1 | Region "/myRegion" created on "server1"
```

debug

Enable or disable debugging output in `gfsh`.

Availability: Online or offline.

Syntax:

```
debug --state=value
```

Name	Description	Default Value
--state	Whether to turn debugging ON or OFF. Valid options are: ON, OFF (Case insensitive)	OFF

Table 1. Debug Parameters

Example Commands:

```
debug --state=off
debug --state=on
```

Sample Output:

```
gfsh>debug --state=on
Debug is on
```

define index

Define an index that can be used when executing queries. Then, you can execute a single command to create multiple indexes all at once using `create defined indexes`.

Availability: Online or offline.

Syntax:

```
define index --name=value --expression=value --region=value [--type=value]
```

Name	Description	Default Value
--name	*Required.* Name of the index to define.	
--expression	*Required.* Field of the region values that are referenced by the index.	
--region	*Required.* Name/Path of the region which corresponds to the “from” clause in a query.	
--type	Type of the index. Valid values are: range, key and hash.	range

Table 1. Define Index Parameters

Example Commands:

```
gfsh> define index --name=myIndex1 --expression=exp1 --region=/exampleRegion

gfsh> define index --name=myIndex2 --expression=”c.exp2” --region="/exampleRegion e,
e.collection1 c"

gfsh> define index --name=myIndex3 --expression=exp3 --region=/exampleRegion --type=ha
sh

//then to create the indexes, execute:

gfsh> create defined indexes
```

Sample Output:

```
gfsh>define index --name=myIndex1 --expression=exp1 --region=/exampleRegion
Index successfully defined with following details
Name       : myIndex1
```

```
Expression : exp1
RegionPath : /exampleRegion
```

deploy

Deploy JAR-packaged applications to a member or members.

Only one of either `--jars` or `--dir` may be specified.

Availability: Online. You must be connected in `gfsh` to a JMX Manager member to use this command.

Syntax:

```
deploy [--groups=value(,value)*] [--jars=value(,value)*] [--dir=value]
```

Name	Description
--groups	Group(s) to which the specified JARs will be deployed. If this option is not specified, the deployment will occur on all members.
--jars	Path(s) of the JAR(s) to deploy.
--dir	Directory from which to deploy the JARs.

Table 1. Deploy Parameters

Example Commands:

```
deploy --jars=group1_functions.jar --groups=Group1

deploy --dir=libs/group1-libs --groups=Group2
```

Sample Output:

```
gfsh> deploy --jars=group1_functions.jar --groups=Group1

 Member   |     Deployed JAR     |                 Deployed JAR Location
--------- | -------------------- | ---------------------------------------------------
datanode1 | group1_functions.jar | /usr/local/gemfire/deploy/GF#group1_functions.jar#1
datanode2 | group1_functions.jar | /usr/local/gemfire/deploy/GF#group1_functions.jar#1


gfsh> deploy --dir=libs/group1-libs --groups=Group2

Deploying files: group2_functions.jar, group2_dependencies.jar
Total file size is: 0.64MB

Continue?  (Y/n): Y

 Member   |      Deployed JAR       |             Deployed JAR Location
--------- | ----------------------- | ---------------------------------------------
datanode3 | group2_functions.jar    | /usr/local/gemfire/deploy/GF#group2_functions.ja
r#1
datanode3 | group2_dependencies.jar | /usr/local/gemfire/deploy/GF#group2_dependencie
s.jar#1
datanode4 | group2_functions.jar    | /usr/local/gemfire/deploy/GF#group2_functions.ja
r#1
```

```
datanode4 | group2_dependencies.jar | /usr/local/gemfire/deploy/GF#group2_dependencie
s.jar#1
```

describe

Display details of a member's configuration, shell connection, disk-stores, members, or regions.

- **describe client**

  Displays details about a specified client.

- **describe config**

  Display the configuration of a member.

- **describe connection**

  Display connection information details.

- **describe disk-store**

  Display information about a member's disk store.

- **describe jndi-binding**

  Display information about the configuration of a JNDI binding.

- **describe lucene index**

  Display information about a Lucene index.

- **describe member**

  Display details of a member with given name/id.

- **describe offline-disk-store**

  Display information about an offline member's disk store.

- **describe query-service**

  Display information about the configuration of the query service.

- **describe region**

  Display the attributes and key information of a region.

describe client

Displays details about a specified client.

Availability: Online. You must be connected in `gfsh` to a JMX Manager member to use this command.

Syntax:

```
describe client --clientID=value
```

Parameters, describe client:

Name	Description
--clientID	*Required.* ID of the client. To find a client ID, you can use the `list clients` command to display a list of connected clients and their IDs.

Example Commands:

```
describe client --clientID=192.0.2.0(4987:loner):58922:7b3398cf
```

Sample Output:

```
gfsh>describe client --clientID=192.0.2.0(4987:loner):58922:7b3398cf
-------------------------------------------------------------------
Primary Servers                 : 192.0.2.0(server1:5764)<v1>:15189
Secondary Servers               : 192.0.2.0(server2:5891)<v2>:39082
CPU                             : 0
Number of Cache Listner Calls   : 0
Number of Gets                  : 0
Number of Misses                : 0
Number of Puts                  : 0
Number of Threads               : 0
Process CPU Time (nanoseconds)  : 0
Queue size                      : 1
UP Time (seconds)               : 67
Is Durable                      : No
```

describe config

Display the configuration of a member.

Availability: Online. You must be connected in `gfsh` to a JMX Manager member to use this command.

Syntax:

```
describe config --member=value [--hide-defaults(=value)?]
```

Parameters, describe config:

Name	Description	Default Value
--member	Name or ID of a member whose configuration is to be shown.	
--hide-defaults	Whether to hide configuration information for properties with the default value.	true

Example Commands:

```
describe config --member=Member1;
```

Sample Output:

```
gfsh>describe config --member=server1
Configuration of member : "server1"


JVM command line arguments
```

```
-----------------------------------
-Dgemfire.mcast-port=0
-Dgemfire.locators=localhost[10334]

GemFire properties defined using the API
................................................
log-file                                 : vf.gf.server.log
name                                     : server1

GemFire properties defined at the runtime
................................................
log-level                                 : finest
statistic-sampling-enabled                : true

Cache attributes
................................................
is-server        : true

Cache-server attributes
 . bind-address         : localhost
```

describe connection

Display connection information details.

Availability: Online. You must be connected in `gfsh` to a JMX Manager member to use this command.

Syntax:

```
describe connection
```

Example Commands:

```
describe connection
```

Sample Output:

```
gfsh>describe connection
Connection Endpoints
--------------------
GemFireUser[1099]
```

describe disk-store

Display information about a member's disk store.

Availability: Online. You must be connected in `gfsh` to a JMX Manager member to use this command.

Syntax:

```
describe disk-store --member=value --name=value
```

Parameters, describe disk-store:

Name	Description
--member	*Required.* Name/ID of the member with the disk store to be described.
--name	*Required.* Name of the disk store to be described.

Example Commands:

```
describe disk-store --member=server1 --name=DiskStore1
```

Sample Output:

```
gfsh>describe disk-store --name=disk1 --member=server1
Disk Store ID                      : a531bc7b-5188-4510-85d7-de7de30c6671
Disk Store Name                    : disk1
Member ID                          : ubuntu(server1:7467)<v1>:35249
Member Name                        : server1
Allow Force Compaction             : No
Auto Compaction                    : Yes
Compaction Threshold               : 50
Max Oplog Size                     : 1024
Queue Size                         : 0
Time Interval                      : 1000
Write Buffer Size                  : 32768
Disk Usage Warning Percentage      : 90
Disk Usage Critical Percentage     : 99
PDX Serialization Meta-Data Stored : No


        Disk Directory          | Size
------------------------------- | ----------
/home/user/server1/DiskStore1 | 2147483647
```

describe jndi-binding

Print the configuration information that describes a JDBC connection.

Availability: Online. You must be connected in `gfsh` to a JMX Manager member to use this command.

Syntax:

```
describe jndi-binding --name=value
```

Parameters, describe jndi-binding:

Name	Description
--name	*Required.* Name of the JNDI binding to be described.

Example Commands:

```
describe jndi-binding --name=jndi1
```

Sample Output:

```
gfsh>describe jndi-binding --name=jndi1
    Property      | Value
----------------- | -----------------------------------
type              | SimpleDataSource
jndi-name         | jndi1
jdbc-driver-class | org.apache.derby.jdbc.EmbeddedDriver
user-name         |
connection-url    | jdbc:derby:newDB
```

describe lucene index

Describe a Lucene index.

See also create lucene index, destroy lucene index, list lucene indexes and search lucene.

Availability: Online. You must be connected in `gfsh` to a JMX Manager member to use this command.

Syntax:

```
describe lucene index --name=value --region=value
```

Parameters, describe lucene index:

Name	Description
--name	*Required.* Name of the Lucene index to describe
--region	*Required.* Name and path of the region in which the Lucene index exists

Example Commands:

```
gfsh>describe lucene index --name=personIndex --region=/Person
```

Sample Output:

```
gfsh>describe lucene index --name=personIndex --region=/Person
Index Name  | Region Path |                 Indexed Fields                 | Field Ana
lyzer |   Status    | Query Executions | Updates | Commits | Documents
----------- | ----------- | ---------------------------------------------- | ---------
----- | ----------- | ---------------- | ------- | ------- | ---------
personIndex | /Person     | [name, email, address, streetAddress, revenue] | {}
| Initialized | 339              | 1008    | 962     | 1004

gfsh>describe lucene index --name=analyzerIndex --region=/Person
 Index Name    | Region Path |     Indexed Fields     |            Field Analyzer
|   Status     | Query Executions | Updates | Commits | Documents
------------- | ----------- | ---------------------- | -------------------------------
------ | ----------- | ---------------- | ------- | ------- | ---------
analyzerIndex | /Person     | [address, name, email] | {address=MyCharacterAnalyzer, e
mail.. | Initialized | 1695             | 1008    | 962     | 1004
```

describe member

Display details of a member with given name/id.

Availability: Online. You must be connected in `gfsh` to a JMX Manager member to use this command.

Syntax:

```
describe member --name=value
```

Parameters, describe member:

Name	Description
--name	*Required*. Display information about a member, including name, ID, groups, regions, etc.

Example Commands:

```
describe member --name=server1
```

Sample Output:

```
gfsh>describe member --name=server1
Name         : server1
Id           : GemFireUser(server1:240)<v1>:64871
Host         : 192.0.2.0
Regions      : region4
region5
region3
region2
region1

PID          : 240
Groups       :
Used Heap    : 5M
Max Heap     : 123M
Working Dir  : c:\PivotalGemFire70\Latest\server1
Log file     : C:\PivotalGemFire70\Latest\server1\vf.gf.server.log
Locators     : localhost[10334]

Server Bind         : localhost
Server Port         : 40404
Running             : true
Client Connections : 0
```

describe offline-disk-store

Display information about an offline member's disk store.

Availability: Online. You must be connected in `gfsh` to a JMX Manager member to use this command.

Syntax:

```
describe offline-disk-store --name=value --disk-dirs=value(,value)* [--pdx=value] [--r
egion=value]
```

Parameters, describe offline-disk-store:

Name	Description
--name	*Required*. Name of the disk store to be described.
--disk-dirs	*Required*. Directory that contains the disk store files.
--pdx	If set (or set to true), display all the pdx types stored in the disk store.
--region	Name and path of the region in the disk store to be described.

Example Commands:

```
describe offline-disk-store --name=DiskStore1\
 --disk-dirs=/home/username/gemfire/mydiskStore1Dir

describe offline-disk-store --name=DiskStore1 --disk-dirs=/DiskDir1  --pdx=true
```

Sample Output:

```
gfsh>describe offline-disk-store --name=DiskStore1 --disk-dirs=/DiskDir1 --pdx=true
Regions in the disk store:
  /PdxTypes: -lru=none -concurrencyLevel=16 -initialCapacity=16 -loadFactor=0.75 -comp
ressor=none -statisticsEnabled=false
  /Region1: -lru=none -concurrencyLevel=16 -initialCapacity=16 -loadFactor=0.75 -compr
essor=none -statisticsEnabled=false
PDX Types:
  com.app.data.PositionPdx: id=1
    long avg20DaysVol;
    String bondRating;
    double convRatio;
    String country;
    double delta;
    long industry;
    long issuer;
    double mktValue;
    double qty;
    String secId; // identity
    String secIdIndexed;
    String secLinks;
    double sharesOutstanding;
    String underlyer;
    long volatility;
    int pid;
    int portfolioId;
  com.app.data.StockPdx: id=2
    int ID; // identity
    String pkid;
    Object position1;
    Object position2;
    Object positions;
    Object collectionHolderMap;
    String type;
    String status;
    String[] names;
    String description;
    long createTime;
    Object[] position3;
    Object aDay;
    Date date;
```

```
PDX Enums:
  com.app.data.StockPdx$Day.Monday
```

describe query-service

Print the information that describes the current configuration of the query service.

Availability: Online. You must be connected in `gfsh` to a JMX Manager member to use this command.

Syntax:

```
describe query-service
```

Example Commands:

```
describe query-service
```

Sample Output:

```
gfsh>describe query-service
Method Authorizer Class : org.apache.geode.cache.query.security.JavaBeanAccessorMethod
Authorizer
```

describe region

Display the attributes and key information of a region.

Availability: Online. You must be connected in `gfsh` to a JMX Manager member to use this command.

Syntax:

```
describe region --name=value
```

Parameters, describe region:

Name	Description
--name	*Required.* Name/Path of the region to be described.

Example Commands:

```
describe region --name=region1
```

Sample Output:

```
gfsh>describe region --name=Region1
..........................................................
Name : Region1
Data Policy : persistent replicate
Hosting Members : server-5
server-4
server-3
```

```
server-2

Non-Default Attributes Shared By Hosting Members

Type   | Name            | Value
------ | --------------- | --------------------
Region | data-policy     | PERSISTENT_REPLICATE
       | disk-store-name | DiskStore1
       | size            | 0
       | scope           | distributed-ack

............................................................
Name             : Region1
Data Policy      : empty
Accessor Members : server-1

Non-Default Attributes Shared By Accessor Members

 Type  |    Name     | Value
------ | ----------- | ---------------
Region | data-policy | EMPTY
       | size        | 0
       | scope       | distributed-ack
```

destroy

Delete or unregister functions, remove indexes, disk stores and regions.

- **destroy async-event-queue**

  Destroy an asynchronous event queue.

- **destroy disk-store**

  Delete a disk store and all files on disk used by the disk store.

- **destroy function**

  Destroy or unregister a function.

- **destroy gateway-receiver**

  Destroy a gateway receiver.

- **destroy gateway-sender**

  Destroy a gateway sender.

- **destroy index**

  Destroy or remove the specified index.

- **destroy jndi-binding**

  Destroy the specified JNDI binding.

- **destroy lucene index**

  Destroy or remove the specified Lucene index.

- **destroy region**

Destroy or remove a region.

destroy async-event-queue

Destroy an asynchronous event queue.

Availability: Online. You must be connected in `gfsh` to a JMX Manager member to use this command.

Syntax:

```
destroy async-event-queue --id=value [--groups=value(,value)*] [--if-exists=value]
```

Parameters, destroy async-event-queue:

Name	Description
--id	*Required.* ID of the async event queue to be deleted.
--groups	Group(s) of members on which the async event queue will be destroyed. If no group is specified, the queue is destroyed on all members.
--if-exists	If the specified async event queue does not exist, gfsh responds with a message to that effect. If this parameter is true, the response is prefixed with the label "Skipping: ". Useful for scripted tests. Default (if the parameter is not specified): false. Default (if the parameter is specified without value): true.

Example Commands:

```
destroy async-event-queue --id=myAsyncEventQueue
```

destroy disk-store

Delete a disk store and all files on disk used by the disk store. Data for closed regions that previously used this disk store are lost.

Availability: Online. You must be connected in `gfsh` to a JMX Manager member to use this command.

Syntax:

```
destroy disk-store --name=value [--groups=value(,value)*] [--if-exists=value]
```

Parameters, destroy disk-store:

Name	Description
--name	*Required.* Name of the disk store to be deleted.
--groups	Group(s) of members on which the disk store will be destroyed. If no group is specified, the disk store is destroyed on all members.
--if-exists	If the specified disk store does not exist, gfsh responds with a message to that effect. If this parameter is true, the response is prefixed with the label "Skipping: ". Useful for scripted tests. Default (if the parameter is not specified): false. Default (if the parameter is specified without value): true.

Example Commands:

```
destroy disk-store --name=store1
```

Sample Output:

```
gfsh>destroy disk-store --name=store1
Member  | Result
------- | -------
server1 | Success
```

destroy function

Destroy or unregister a function.

The default is for the function to be unregistered from all members.

Availability: Online. You must be connected in `gfsh` to a JMX Manager member to use this command.

Syntax:

```
destroy function --id=value [--groups=value(,value)*] [--member=value]
```

Parameters, destroy function:

Name	Description
--id	*Required.* Unique function identifier. Use the `list functions` command to obtain the ID.
--groups	One or more groups of members from which this function will be unregistered.
--member	Name or ID of the member from which this function will be unregistered.

Example Commands:

```
(1) destroy function --id=InterestCalculations
(2) destroy function --id=InterestCalculations --member=server1
(3) destroy function --id=InterestCalculations --group=Group1
```

destroy gateway-receiver

Destroy the gateway receiver on one or more members. Since there can be only one gateway receiver per member, specifying the member specifies which gateway receiver to destroy.

Availability: Online. You must be connected in `gfsh` to a JMX Manager member to use this command.

Syntax:

```
destroy gateway-receiver [--group=value(,value)*]
  [--member=value(,value)*] [--if-exists(=value)?]
```

Parameters, destroy gateway-receiver:

Name	Description
--group	One or more groups of members on which the gateway receiver will be destroyed.
--members	Name or ID of each member on which the gateway receiver will be destroyed.
--if-exists	If this option is true and the gateway receiver does not exist, gfsh responds with a message prefixed with the label "Skipping: ". If this option is false and the gateway receiver does not exist, an error message is output. Default (if this option is not specified): false. Default (if this option is specified without a value): true.

Example Commands:

```
destroy gateway-receiver --member=server1,server2
```

destroy gateway-sender

Destroy a gateway sender that is no longer used by a region.

The default is for the gateway sender to be destroyed on all members.

No region may be attached to the gateway sender to be destroyed. If a region is still attached, the system issues an error message similar to:

```
ERROR: The GatewaySender ParallelGatewaySender{id=ln,remoteDsId=2,isRunning =false}
could not be destroyed as it is still used by region(s).
```

Remove the gateway sender from the region with a command similar to:

```
gfsh>alter region --name=regionA --gateway-sender-id=""
```

Availability: Online. You must be connected in `gfsh` to a JMX Manager member to use this command.

Syntax:

```
destroy gateway-sender --id=value [--groups=value(,value)*]
  [--members=value(,value)*] [--if-exists=value]
```

Parameters, destroy gateway-sender:

Name	Description
--id	*Required.* Unique gateway sender identifier. Use the `list gateways` command to obtain the ID.
--groups	One or more groups of members from which this gateway sender will be destroyed.
--members	Name or ID of the member(s) from which this gateway sender will be destroyed.
--if-exists	If the specified gateway sender does not exist, gfsh responds with a message to that effect. If this parameter is true, the response is prefixed with the label "Skipping: ". Useful for scripted tests. Default (if the parameter is not specified): false. Default (if the parameter is specified without value): true.

Example Commands:

```
destroy gateway-sender --id=SiteASender
```

destroy index

Destroy or remove the specified index.

Availability: Online. You must be connected in `gfsh` to a JMX Manager member to use this command.

Syntax:

```
destroy index [--name=value] [--region=value] [--members=value(,value)*]
[--groups=value(,value)*] [--if-exists=value]
```

Note: You must specify at least one of the parameter options. If you enter `destroy index` without any parameters, the command will ask you to specify at least one option.

Parameters, destroy index:

Name	Description
--name	Name of the index to be removed.
--members	Id of the member(s) on which index is to be removed.
--region	Name of the region from which an index or all indexes are to be destroyed.
--groups	The index will be removed on all the members in the group(s).
--if-exists	If the specified index does not exist, gfsh responds with a message to that effect. If this parameter is true, the response is prefixed with the label "Skipping: ". Useful for scripted tests. Default (if the parameter is not specified): false. Default (if the parameter is specified without value): true.

Example Commands:

```
destroy index --members=server2
destroy index --name=MyKeyIndex
```

destroy jndi-binding

Destroy a specified JNDI binding that holds the configuration for an XA data source.

Availability: Online. You must be connected in `gfsh` to a JMX Manager member to use this command.

Syntax:

```
destroy jndi-binding --name=value [--if-exists=value]
```

Parameters, destroy jndi-binding:

Name	Description
--name	*Required.* Name of the JNDI binding to be destroyed.
--if-exists	Skip the destroy operation when the specified JNDI binding does not exist. Without this option, an error results from the specification of a JNDI binding that does not exist. Default (if the parameter is not specified): false. Default (if the parameter is specified without value): true.

Example Command:

```
destroy jndi-binding --name=jndi1
```

destroy lucene index

Destroy or remove the specified Lucene index.

See also create lucene index, describe lucene index, list lucene indexes and search lucene.

Availability: Online. You must be connected in `gfsh` to a JMX Manager member to use this command.

Syntax:

```
destroy lucene index --region=value [--name=value]
```

Parameters, destroy lucene index:

Name	Description
--region	*Required.* Name of the region from which indexes are to be removed. If no `--name` option is specified, all indexes associated with the region are destroyed.
--name	Name of the index to be removed.

Example Commands:

```
destroy lucene index --region=region1
destroy lucene index --region=region1 --name=MyKeyIndex
```

destroy region

Destroy or remove a region.

Availability: Online. You must be connected in `gfsh` to a JMX Manager member to use this command.

Syntax:

```
destroy region --name=value [--if-exists=value]
```

Parameters, destroy region:

Name	Description
--name	*Required.* Name and path of the region to be removed.

Name	Description
--if-exists	If the specified region does not exist, gfsh responds with a message to that effect. If this parameter is true, the response is prefixed with the label "Skipping: ". Useful for scripted tests. Default (if the parameter is not specified): false. Default (if the parameter is specified without value): true.

Example Commands:

```
destroy region --name=region4
destroy region --name=/region1/subregion1
```

Sample Output:

```
gfsh>destroy region --name=region1
"region1"  destroyed successfully.
```

disconnect

Close any active connection(s).

Availability: Online. You must be connected in `gfsh` to a JMX Manager member to use this command.

Syntax:

```
disconnect
```

Example Commands:

```
disconnect
```

Sample Output:

```
gfsh>disconnect
Disconnecting from: Locator1[1099]
Disconnected from : Locator1[1099]
```

Error Messages:

```
Error occurred while disconnecting: {0}
```

```
Not connected!
```

echo

Echo the given text, which may include system and user variables.

The command can also echo gfsh environment properties (using ‘set variable’ command) if variable name is pre-pended with ‘$’ - like UNIX.

See Useful gfsh Shell Variables for a list of gfsh environment variables.

Availability: Online or offline.

Syntax:

```
echo [--string=value]
```

Name	Description
--string	String to be echoed. For example, `SYS_USER variable is set to ${SYS_USER}`.

Table 1. Echo Parameters

Example Commands:

```
echo --string="Hello World!"
echo --string="Hello World! This is ${SYS_USER}"
echo --string=${APP_FETCH_SIZE}
```

To see all the variable set in the shell:

```
echo --string=$*
```

Sample Output:

```
gfsh>echo --string=${SYS_JAVA_VERSION}
Post substitution: echo --string=1.8.0_60
1.8.0_60
```

execute function

Execute functions on members or regions.

execute function

Execute the function with the specified ID. By default, the function executes on all members.

Availability: Online. You must be connected in `gfsh` to a JMX Manager member to use this command.

Syntax:

```
execute function --id=value [--groups=value(,value)*]
[--members=value(,value)*] [--region=value]
[--arguments=value] [--result-collector=value] [--filter=value]
```

--id	*Required.* ID of the function to execute.
--groups	One or more groups of members on which this function should be executed.
--members	Name/ID of the member(s) on which the function will be executed.
--region	Region on which the data dependent function will be executed.
--arguments	Arguments to the function in comma-separated string format.

--result-collector	Fully qualified class name of the `ResultCollector` to instantiate for gathering results.
--filter	Key list which causes the function to only be executed on members which have entries with these keys.

Table 1. Execute Function Parameters

Example Commands:

```
execute function --id=InterestCalculations --region=/InterestRegion
execute function --id=InterestCalculations --members=server1
execute function --id=InterestCalculations --groups=Group1
```

Note: The `execute function` passes the value of `--arguments` as a String Array to any executed function. This differs from the behavior of `FunctionExecutionService` API. `FunctionExecutionService` API passes the value of the argument as an Object.

exit

Exit the `gfsh` shell. You can also use `quit` to exit the shell.

Exits the gfsh shell and returns to the OS shell.

Availability: Online or offline.

Syntax:

```
exit
```

Example Commands:

```
exit
```

export

Export configurations, data, logs and stack-traces.

- **export cluster-configuration**

  Export a cluster configuration ZIP file that contains the `cache.xml` files, `gemfire.properties` files, and application JAR files needed to configure and operate a cluster.

- **export config**

  Export configuration properties for a member or members.

- **export data**

  Export user data from a region to a file.

- **export logs**

  Export logs to a given directory.

- **export offline-disk-store**

  Export region data from an offline disk store into gemfire snapshot files.
- **export stack-traces**

  Export the stack trace for a member or members.

export cluster-configuration

Exports a single XML file or a ZIP file with cluster configuration that contains the `cache.xml` files, `gemfire.properties` files, and application JAR files needed to configure and operate a cluster.

When neither a file name nor a ZIP file name is specified, the cluster configuration is written to standard output.

Availability: Online. You must be connected in `gfsh` to a JMX Manager member to use this command.

See Overview of the Cluster Configuration Service.

Syntax:

```
export cluster-configuration [--group(=value)?] [--xml-file=value]
 [--zip-file-name=value]
```

Export Cluster-Configuration Parameters:

Name	Description	Default Value
--group	Export the configuration for the specified server group. When no group is specified, use the default `cluster` group.	
--xml-file	Filename to contain the exported cluster configuration. May also include an absolute or relative path. Only specify one of `--xml-file` or `--zip-file-name`.	
--zip-file-name	Filename of the ZIP file to contain the exported cluster configuration. May also include an absolute or relative path. Only specify one of `--xml-file` or `--zip-file-name`.	

Example Commands:

```
gfsh>export cluster-configuration --zip-file-name=/group/shared-configs/devClusterConf
ig.zip
gfsh>export cluster-configuration --zip-file-name=my-configs/myClusterConfig.zip
gfsh>export cluster-configuration --zip-file-name=myClusterConfig.zip
gfsh>export cluster-configuration --xml-file=Cluster3Config.xml
```

Sample Output:

```
gfsh>export cluster-configuration --zip-file-name=mySharedConfig.zip
Downloading cluster configuration : /home/username/gemfire/mySharedConfig.zip
```

export config

Export configuration properties for a member or members.

If you do not specify any parameters, all member configurations will be exported.

Availability: Online. You must be connected in `gfsh` to a JMX Manager member to use this command.

Syntax:

```
export config [--members=value(,value)*] [--groups=value(,value)*]
[--dir=value]
```

Export Config Parameters:

Name	Description
--members	Name/Id of the member(s) whose configuration will be exported.
--groups	Group(s) of members whose configuration will be exported.
--dir	Directory to which the exported configuration files will be written.

Example Commands:

```
export config
export config --members=member1
```

Sample Output:

```
gfsh>export config --members=member1
Downloading Cache XML file: c:\PivotalGemFire\Latest\.\member1-cache.xml
Downloading properties file: c:\PivotalGemFire\Latest\.\member1-gf.properties
```

export data

Export user data from a region to a file or files.

Availability: Online. You must be connected in `gfsh` to a JMX Manager member to use this command.

Syntax:

To export data from any region, replicated or partitioned, sequentially to a single file, use the `--file` option:

```
export data --region=value --member=value --file=value
```

In this scenario, data from replicated and partitioned regions is exported sequentially to a single file on the specified member.

In the case of partitioned regions, you can speed up the export process by using the `--parallel` option:

```
export data  --parallel --region=value --member=value --dir=value
```

In this scenario, partitioned region data is exported simultaneously on all hosting nodes to directories local to those respective nodes.

- Parallel export applies only to partitioned regions.
- The `--file` and `--dir` options are mutually exclusive; `--file` can be used only for serial export, `--dir` can be used for both serial and parallel exports.

Export Data Parameters:

Name	Description
--region	*Required.* Region from which data is to be exported.
--member	*Required.* Name/Id of a member that hosts the region. In a serial export, all data is exported to the specified file on the host where the member is running. In a parallel export, data from a partitioned region partially hosted on this member is exported for each partition to files on the nodes that host those partitions.
--file	File to which the exported data is to be written. The file must have an extension of `.gfd`. Cannot be specified at the same time as --dir, cannot be used with --parallel.
--dir	Directory to which the exported data is to be written. Required if --parallel is true. Cannot be specified at the same time as --file.
--parallel	Export local data on each node to a directory on that machine. Available for partitioned regions only.

Example Commands:

```
export data --region=region2 --file=region2_20121001.gfd --member=server2
```

Sample Output:

```
 gfsh>export data --region=region2 --file=region2_20121001.gfd --member=server1
Data succesfully exported from region : region2 to file : C:\PivotalGemFire\
Latest\server1\region2_20121001.gfd on host : 192.0.2.0
```

export logs

Export logs to a given directory.

All files that have logs in the specified time range will be exported. If no time range is specified, all logs will be exported.

The `--dir` parameter specifies a local directory to which log files will be written. This is used only when you are exporting logs using an http connection. If executed over http, the zip archive will be saved in the specified directory on the user's client machine. If not specified, logs are written to the location specified by the `user.dir` system property. When the command is executed over JMX, logs will be saved as `exportedLogs_xxx.zip` in the connected locator's working directory.

Availability: Online. You must be connected in `gfsh` to a JMX Manager member to use this command.

Syntax:

```
export logs [--dir=value] [--groups=value(,value)*] [--members=value(,value)*]
[--log-level=value] [--only-log-level=value] [--merge-log=value]
[--start-time=value] [--end-time=value] [logs-only(=value)?]
[--stats-only(=value)?] [--file-size-limit(=value)?]
```

Export Logs Parameters:

Name	Description	Default Value
--dir	Local directory to which log files will be written when logs are exported using an http connection. Ignored when the command is executed over JMX.	
--groups	Group(s) of members whose log files will be exported.	
--members	Name/ID of the member(s) whose log files will be exported.	
--log-level	Minimum level of log entries to export. Valid values are: `OFF`, `FATAL`, `ERROR`, `WARN`, `INFO`, `DEBUG`, `TRACE`, and `ALL`.	`INFO`
--only-log-level	Whether to only include only entries that exactly match the --log-level specified.	false
--merge-log	Whether to merge logs after exporting to the target directory (deprecated).	false
--start-time	Log entries that occurred after this time will be exported. Format: `yyyy/MM/dd/HH/mm/ss/SSS/z` OR `yyyy/MM/dd`. When using the `yyyy/MM/dd/HH/mm/ss/SSS/z` format, the time zone, denoted `z`, should be specified as either a 3-letter time zone such as `PST` or as an offset to GMT/UTC such as `GMT+08:00`.	no limit
--end-time	Log entries that occurred before this time will be exported. Format: `yyyy/MM/dd/HH/mm/ss/SSS/z` OR `yyyy/MM/dd`. When using the `yyyy/MM/dd/HH/mm/ss/SSS/z` format, the time zone, denoted `z`, should be specified as either a 3-letter time zone such as `PST` or as an offset to GMT/UTC such as `GMT+08:00`. An end time specified by only a date implements a time of `00:00`. This exports logs written up until `23:59:59.999` on the date prior to the one specified.	no limit
--logs-only	Whether to export only logs (not statistics)	If parameter not specified: false. If parameter specified without a value: true
--stats-only	Whether to export only statistics (not logs)	If parameter not specified: false. If parameter specified without a value: true
--file-size-limit	Limits total unzipped size of the exported files. Specify 0 (zero) for no limit. Value is in megabytes by default or [k,m,g,t] may be specified.	If parameter not specified: 100m. If parameter specified without a value: 0

Example commands, showing output:

```
gfsh>export logs --dir=data/logs
Logs exported to the connected member's file system: /my-locator/data/logs/exportedLog
s_1489513007261.zip
```

```
gfsh>export logs --start-time=2020/12/14/12/00/00/000/GMT-08:00 --end-time=2020/12/27
--dir=data/logs
Start time parsed as 2020-12-14T12:00 PST
End time parsed as 2020-12-27T00:00 PST
Logs exported to the connected member's file system: /my-locator/data/logs/exportedLog
s_1608165308676.zip
```

```
gfsh>export logs --dir=data/logs --file-size-limit=1k
Estimated exported logs expanded file size = 95599, file-size-limit = 1024.
To disable exported logs file size check use option "--file-size-limit=0".

gfsh>export logs --dir=data/logs --file-size-limit=99k
Logs exported to the connected member's file system: /my-locator/data/logs/exportedLog
s_1489513007261.zip
```

export offline-disk-store

Export region data from an offline disk store into gemfire snapshot files.

Availability: Online or offline.

Syntax:

```
export offline-disk-store --name=value --disk-dirs=value(,value)* --dir=value
```

Export Offline-Disk-Store Parameters:

Name	Description
--name	*Required.* Name of the disk store to be exported.
--disk-dirs	Directories which contain the disk store files.
--dir	Directory to export the snapshot files to.

Example Commands:

```
 export offline-disk-store --name= DiskStore1 \
--disk-dirs=/home/username/gemfire/mydiskStore1Dir --dir=/home/username/gemfire/export
```

export stack-traces

Export the stack trace for one or more servers.

Availability: Online. You must be connected in `gfsh` to a JMX Manager member to use this command.

Syntax:

```
export stack-traces [--members=value(,value)*] [--groups=value(,value)*]
[--file=value] [--abort-if-file-exists(=value)?]
```

Export Stack-Traces Parameters:

Name	Description
--members	Name or ID of the member(s) whose stack traces will be exported.
--groups	Group(s) of members whose stack traces will be exported.
--file	File name to which the stack traces will be written. When not specified, the file name will be the string “stacktrace_” followed by the current time in milliseconds.
--abort-if-file-exists	When true, abort the command if the file already exists in the locator’s directory. If this option is not specified, the default value is false.

Example Commands:

```
export stack-traces --file=stack.txt --abort-if-file-exists=true
```

Sample Output:

```
gfsh>export stack-traces
stack-trace(s) exported to file: /data/stacktrace_1612316330340
On host : cluster-locator-0.cluster-locator.system.svc.cluster.local
```

gc

Force GC (Garbage Collection) on a member or members.

The default is for garbage collection to occur on all caching members.

Availability: Online. You must be connected in `gfsh` to a JMX Manager member to use this command.

Syntax:

```
gc [--groups=value(,value)*] [--member=value]
```

Name	Description
--groups	One or more group(s) of members on which garbage collection will be forced.
--member	Name/ID of the member on which garbage collection will be forced.

Table 1. GC Parameters

Example Commands:

```
gc --member=server1
gc --groups=Group1
gc
```

Sample Output:

```
gfsh>gc
Sucessfully executed GC
```

get

Display an entry in a region.

Availability: Online. You must be connected in `gfsh` to a JMX Manager member to use this command.

Syntax:

```
get --key=value --region=value [--key-class=value] [--value-class=value]
```

Name	Description	Default Value
--key	*Required.* String or JSON text from which to create the key. For example: "`James`", "`100L`" and "`('id': '134s')`".	
--region	*Required.* Region from which to get the entry.	
--key-class	Fully qualified class name of the key's type.	The default is the key constraint for the current region or String.
--value-class	Fully qualified class name of the value's type.	The default is the value constraint for the current region or String.
--load-on-cache-miss	Explicitly enables or disables the use of any registered CacheLoaders on the specified Region when retrieving a value for the specified Key on Cache misses.	true (enabled)

Table 1. Get Parameters

Example Commands:

```
get --key=('id':'133abg124') --region=region1

// Retrieving when key type is a wrapper(primitive)/String
get --key=('133abg124') --region=/region1/region12 --value-class=data.ProfileDetails

get --key=('100L') --region=/region1/region12 --value-class=data.ProfileDetails
--key-class=java.lang.Long
```

Sample Output:

```
gfsh>get --key=('123') --region=region1
Result      : true
Key Class   : java.lang.String
Key         : ('123')
Value Class : java.lang.String
Value       : ABC
```

help

Display syntax and usage information for all the available commands.

Typing help without a command as an argument lists all available commands.

Availability: Online or offline.

Syntax:

```
help [command]
```

Examples Commands:

```
help
help rebalance
```

Sample Output:

```
gfsh>help rebalance
NAME
        rebalance
IS AVAILABLE
        true
SYNOPSIS
        Rebalance partitioned regions. The default is for all partitioned region
s to be rebalanced.
SYNTAX
        rebalance [--include-region=value(,value)*] [--exclude-region=value(,val
ue)*] [--time-out=value] [--simulate(=value)?]
PARAMETERS
        include-region
                Partitioned regions to be included when rebalancing. Includes ta
ke precedence over excludes.
                Required:false
        exclude-region
                Partitioned regions to be excluded when rebalancing.
                Required:false
        time-out
                Time to wait (in seconds) before GFSH returns to a prompt while
rebalancing continues in the background. The default is to wait for rebalancing
to complete.
                Required:false
                Default if the parameter is not specified:-1
        simulate
                Whether to only simulate rebalancing. The --time-out parameter i
s not available when simulating.
                Required:false
                Default if no value for the parameter is given:true
                Default if the parameter is not specified:false
```

hint

Display information on topics and a list of commands associated with a topic.

Provide hints for a topic or lists all available topics if a topic is not specified.

Availability: Online or offline.

Syntax:

```
hint [topic]
```

Example Commands:

```
hint
hint Server
```

Sample Output:

```
gfsh>hint
Hints are available for the following topics. Use "hint <topic-name>" for a specific
 hint.
Configuration
Data
Debug-Utility
Disk Store
Function Execution
GFSH
Help
JMX
Lifecycle
Locator
Management-Monitoring
Manager
Region
Server
Statistics

gfsh>hint server
A server is GemFire cluster member which holds a GemFire cache. Depending on the
 topology used it can refer to either a system that responds to client requests
or a system that is only a peer to other members.

describe member : Display information about a member, including name, id, groups
, regions, etc.
export logs     : Export the log files for a member or members.
list members    : Display all or a subset of members.
start server    : Start a GemFire Cache Server.
status server   : Display the status of a GemFire Cache Server.
stop server     : Stop a GemFire Cache Server..
```

history

Show or save the command history.

This history can be saved to a file which can also be used as a script later.

A history of commands that have been executed successfully is also logged in the `.geode/.gfsh.history` file within the home directory of the user running gfsh.

Availability: Online or offline.

Syntax:

```
history [--file=<history text file>]
```

Name	Description	Default Value
--file	File to which the history is to be saved.	
--clear	When set to `true`, clears the history of gfsh commands.	false

Table 1. History Parameters

Example Commands:

```
history
history --file=./mycommands.gfsh;
```

Sample Output:

```
gfsh>history --file=./mycommands.gfsh
Wrote successfully to file ./mycommands.gfsh
```

import

You can import an exported cluster configuration to create a new cluster or import data into a region.

- **import cluster-configuration**

  Import a cluster configuration.

- **import data**

  Import user data from a file to a region.

import cluster-configuration

Imports a previously exported cluster configuration from a ZIP file or an XML file. This command is useful when spinning up a new cluster.

- In a given cluster, only one locator needs to perform the import. That locator shares the imported configuration with all other connected locators in the cluster.
- Locators share the imported configuration with data members (servers) when the servers start, or if the servers have been recently started, have no regions defined in them, and have been given no other configuration changes since they started.

To import a cluster configuration, start one or more locators and then run the `gfsh import cluster-configuration` command.

Availability: Online. You must be connected in `gfsh` to a locator to use this command.

Syntax:

```
import cluster-configuration [--action=value] [--group(=value)?]
 [--xml-file=value] [--zip-file-name=value]
```

Import Cluster-Configuration Parameters:

Name	Description
--action	When the value is `APPLY` (the default), the configuration is applied to the running servers that have no configuration. If any servers already have configuration, the command fails. When the value is `STAGE`, the configuration is overwritten and will be used during future server creation; the configuration of currently running servers is not changed.

Name	Description
--group	Do the import for the specified server group. When no group is specified, the `cluster` is implied.
--xml-file	Filename from which to import the cluster configuration. May also include an absolute or relative path. Only specify one of `--xml-file` or `--zip-file-name`.
--zip-file-name	Name of the ZIP file containing the cluster configuration artifacts to be imported. Only specify one of `--xml-file` or `--zip-file-name`.

Example Commands:

```
gfsh>import cluster-configuration --zip-file-name=/home/username/myClusterConfig.zip
gfsh>import cluster-configuration --xml-file=configs/Cluster3Config.xml
```

Sample Output:

```
gfsh>import cluster-configuration --zip-file-name=/home/username/myClusterConfig.zip
Cluster configuration successfully imported
```

import data

Import user data from a file or files to a region.

Availability: Online. You must be connected in `gfsh` to a JMX Manager member to use this command.

Syntax:

If the data was exported sequentially to a single file, use the serial form of the `import` command to import the data:

```
import data --region=value --file=value --member=value [--invoke-callbacks=value]
```

In this scenario, data from replicated and partitioned regions is imported sequentially from a single file on the specified member.

See export data for an explanation of the difference between serial and parallel data exports.

If the data was exported from a partitioned region in parallel format, use the `--parallel` option to import the data:

```
import data --parallel --region=value --member=value --dir=value
  [--invoke-callbacks=value]
```

In this scenario, partitioned region data is imported simultaneously on all hosting nodes from directories local to those respective nodes.

- Parallel import applies only to partitioned regions exported in parallel mode.
- The `--file` and `--dir` options are mutually exclusive; `--file` can be used only for serial import, `--dir` can be used for both serial and parallel imports.

Import Data Parameters:

Name	Description	Default Value
--region	*Required.* Region into which data will be imported.	
--member	*Required.* Name/ID of a member that hosts the region. In a serial import, all data is imported from the specified file on the host where the member is running. In a parallel import, data from a partitioned region partially hosted on this member is imported for each partition from files on the nodes that host those partitions.	
--file	File from which the imported data will be read. The file must have an extension of `.gfd` and must be on the file system of the member (or accessible to that member via NFS) that is being targeted for the import. Cannot be specified at the same time as --dir, cannot be used with --parallel.	
--dir	Directory from which the data is to be imported. Required if --parallel is true. Cannot be specified at the same time as --file. If a given node has no data file in the specified directory, the import operation is silently skipped for that node.	
--invoke-callbacks	Boolean value that, when true, invokes callbacks during the data import.	false

Example Commands:

```
import data --region=region2 --file=/mnt5/region2_20121001.gfd --member=server1
```

list

List existing VMware GemFire resources such as deployed applications, disk-stores, functions, members, servers, and regions.

- **list async-event-queues**

  Display a list of async event queues for all members.

- **list clients**

  Display a list of connected clients.

- **list deployed**

  Display a list of JARs that were deployed to members using the deploy command.

- **list disk-stores**

  List all available disk stores across the VMware GemFire cluster

- **list durable-cqs**

  List durable client CQs associated with the specified durable client id.

- **list functions**

  Display a list of registered functions. The default is to display functions for all members.

- **list gateways**

  Display the gateway senders and receivers for a member or members.

- **list indexes**

  Display the list of indexes created for all members.

- **list jndi-binding**

  List all JNDI bindings, active and configured.
- **list lucene indexes**

  List Lucene indexes created for all members.
- **list members**

  Display all or a subset of members.
- **list regions**

  Display regions of a member or members. If no parameter is specified, all regions in the cluster are listed.

list async-event-queues

Display a list of async event queues for all members.

Availability: Online. You must be connected in `gfsh` to a JMX Manager member to use this command.

Syntax:

```
list async-event-queues
```

Example Commands:

```
list async-event-queues
```

list clients

Display a list of connected clients and the servers to which they connect.

Availability: Online. You must be connected in `gfsh` to a JMX Manager member to use this command.

Syntax:

```
list clients
```

Example Commands:

```
list clients
```

Sample Output:

```
gfsh>list clients

ClientList

          Client Name / ID            | Server Name / ID
------------------------------------ | ----------------------------------------------
------
192.0.2.0(4987:loner):58922:7b3398cf | member=server2,port=53508;  member=server1,port
```

```
=56806
192.0.2.0(5065:loner):39906:a6f598cf | member=server2,port=53508;  member=server1,port
=56806
```

list deployed

Display a list of JARs that were deployed to members using the deploy command.

Availability: Online. You must be connected in `gfsh` to a JMX Manager member to use this command.

Syntax:

```
list deployed [--groups=value(,value)*]
```

Parameters, list deployed:

Name	Description
--groups	Group(s) of members for which deployed JARs will be displayed. If not specified, JARs for all members are displayed.

Example Commands:

```
list deployed
list deployed --groups=Group2
```

Sample Output:

```
gfsh> list deployed  --groups=Group2

 Member   |     Deployed JAR     |                  JAR Location
--------- | -------------------- | ---------------------------------------------------
datanode1 | group1_functions.jar | /usr/local/gemfire/deploy/vf.gf#group1_functions.ja
r#1
datanode2 | group1_functions.jar | /usr/local/gemfire/deploy/vf.gf#group1_functions.ja
r#1
```

Error Messages:

```
No JAR Files Found
```

list disk-stores

List all available disk stores across the VMware GemFire cluster.

The command also lists the configured disk directories and any Regions, Cache Servers, Gateways, PDX Serialization and Async Event Queues using Disk Stores to either overflow and/or persist information to disk. Use the `describe disk-store` command to see the details for a particular Disk Store.

Availability: Online. You must be connected in `gfsh` to a JMX Manager member to use this command.

Syntax:

```
list disk-stores
```

Example Commands:

```
list disk-stores
```

Sample Output:

```
gfsh> list disk-stores

   Member Name   |                   Member Id                  | Disk Store Name |
Disk Store ID
  -------------- | ------------------------------------------| --------------- | -----
-------------------------------
  consumerServer | 192.0.2.0(consumerServer:13825)<v5>:3545  | consumerData    | 4029a
f26-fd82-4997-bd6c-33382cdbb5e9
  consumerServer | 192.0.2.0(consumerServer:13825)<v5>:3545  | observerData    | 7e031
6ad-963c-49b0-9b01-8f59b8d9e29e
  producerServer | 192.0.2.0(producerServer:13826)<v3>:53764 | producerData    | 4670e
4eb-1c50-4465-b418-08ede3d5dbed
```

Error Messages:

```
No Disk Stores Found
```

list durable-cqs

List durable client CQs associated with the specified durable client id.

Availability: Online. You must be connected in `gfsh` to a JMX Manager member to use this command.

Syntax:

```
list durable-cqs --durable-client-id=value
 [--members=value(,value)*] [--groups=value(,value)*]
```

Parameters, list durable-cqs:

Name	Description
--durable-client-id	*Required.* The ID used to identify the durable client.
--members	Name or Id of the member(s) for which the durable client is registered and durable CQs will be displayed.
--groups	Group(s) of members for which the durable client is registered and durable CQs will be displayed.

Example Commands:

```
list durable-cqs --durable-client-id=client1
```

Sample Output:

```
gfsh>list durable-cqs --durable-client-id=client1
member  | durable-cq-name
------- | ---------------
server3 | cq3
        | cq1
        | cq2
server4 | cq3
        | cq1
```

Error Messages:

```
Unable to list durable-cqs for durable-client-id : "client1" due to following reasons.

No client found with client-id : client1
Occurred on members
1.server4
2.server1
3.server3
```

list functions

Display a list of registered functions. The default is to display functions for all members.

Availability: Online. You must be connected in `gfsh` to a JMX Manager member to use this command.

Syntax:

```
list functions [--matches=value] [--groups=value(,value)*]
[--members=value(,value)*]
```

Parameters, list functions:

Name	Description
--matches	Pattern that the function ID must match in order to be included. Uses Java pattern matching rules, not UNIX. For example, to match any character any number of times use ".*" rather than "*".
--groups	Group(s) of members for which functions will be displayed. Use a comma separated list for multiple groups.
--members	Name or ID of the member(s) for which functions will be displayed. Use a comma separated list for multiple members.

Example Commands:

```
list functions
list functions --matches=reconcile.*
```

Sample Output:

```
gfsh> list functions

   Member   |          Function
  --------- | --------------------------
  camelot   | loadDataFromExternalSource
  camelot   | reconcileWeeklyExpenses
  excalibur | loadDataFromExternalSource
  excalibur | reconcileDailyExpenses
```

Example of `list functions` with a “matches” filter:

```
gfsh> list functions --matches=reconcile.*

   Member   |        Function
  --------- | -----------------------
  camelot   | reconcileWeeklyExpenses
  excalibur | reconcileDailyExpenses
```

Error Messages:

```
No Functions Found
```

list gateways

Display the gateway senders and receivers for a member or members.

Availability: Online. You must be connected in `gfsh` to a JMX Manager member to use this command.

Syntax:

```
list gateways [--members=value(,value)*] [--groups=value(,value)*]
```

Parameters, list gateways:

Name	Description
--members	Member(s) whose gateways senders and receiver display.
--groups	Group(s) of members for which Gateway Senders and Receivers will be displayed. Use a comma separated list for multiple groups.
--senders-only	List only gateway senders. This parameter cannot be used together with `--receivers-only`
--receivers-only	List only gateway receivers. This parameter cannot be used together with `--senders-only`.

Example Commands:

```
list gateways
```

Sample Output:

```

gfsh>list gateways

GatewaySender Section

GatewaySender Id |               Member          | Remote Cluster Id |  Type    | Status
| Queued Events | Receiver Location
---------------- | --------------------------- | ----------------- | -------- | ------
```

```
- | ------------- | -----------------
ln              | mymac(ny-1:88641)<v2>:33491 | 2                 | Parallel | Runnin
g | 0            | mymac:5037
ln              | mymac(ny-2:88705)<v3>:29329 | 2                 | Parallel | Runnin
g | 0            | mymac:5064
ln              | mymac(ny-3:88715)<v4>:36808 | 2                 | Parallel | Runnin
g | 0            | mymac:5132
ln              | mymac(ny-4:88724)<v5>:52993 | 2                 | Parallel | Runnin
g | 0            | mymac:5324

GatewayReceiver Section

         Member              | Port | Sender Count | Senders Connected
--------------------------- | ---- | ------------ | ----------------------------------
--------------------------------------------------------------------------------------
mymac(ny-1:88641)<v2>:33491 | 5057 | 9            |["mymac(ln-1:88651)<v2>:48277","mym
ac(ln-4:88681)<v5>:42784","mymac(ln-3:88672)<v4>:43675","mymac(ln-2:88662)<v3>:12796"]
mymac(ny-2:88705)<v3>:29329 | 5082 | 4            |["mymac(ln-1:88651)<v2>:48277","mym
ac(ln-4:88681)<v5>:42784","mymac(ln-3:88672)<v4>:43675"]
mymac(ny-3:88715)<v4>:36808 | 5371 | 4            |["mymac(ln-1:88651)<v2>:48277","mym
ac(ln-4:88681)<v5>:42784","mymac(ln-3:88672)<v4>:43675"]
mymac(ny-4:88724)<v5>:52993 | 5247 | 3            |["mymac(ln-1:88651)<v2>:48277","mym
ac(ln-4:88681)<v5>:42784","mymac(ln-3:88672)<v4>:43675"]
```

list indexes

Display the list of indexes created for all members.

Availability: Online. You must be connected in `gfsh` to a JMX Manager member to use this command.

Syntax:

```
list indexes [--with-stats(=value)?]
```

Parameters, list indexes:

Name	Description	Default Value
--with-stats	Specifies whether statistics should also be displayed.	false

Example Commands:

```
list indexes
list indexes --with-stats
```

Sample Output:

```
gfsh>list indexes
Member Name    |                   Member ID                | Region Path |   Name   |
Type  | Indexed Expression | From Clause
-------------- | ----------------------------------------- | ----------- | -------- |
----- | ------------------ | -----------
consumerServer | 192.0.2.0(consumerServer:13873):6317      | /consumers  | cidIdx   |
KEY   | id                 | /consumers
consumerServer | 192.0.2.0(consumerServer:13873):6317      | /consumers  | cnameIdx |
RANGE | name                | /consumers
producerServer | 192.0.2.0(producerServer:13874):19198     | /producers  | pidIdx   |
RANGE | id                  | /producers
```

Example of ‘list indexes’ with stats printed:

```
gfsh>list indexes --with-stats

Member Name  | Member ID | Region Path |   Name   | Type  | Indexed Expression | From
Clause | Uses | Updates | Update Time | Keys | Values
------------ | --------- | ----------- | -------- | ----- | ------------------ | -----
------ | ---- | ------- | ----------- | ---- | ------
cs...        | 192...    | /consumers  | cidIdx   | KEY   | id                 | /cons
umers  | 2512 | 0       | 0           | 5020 | 5020
cs...        | 192...    | /consumers  | cnameIdx | RANGE | name               | /cons
umers  | 0    | 5020    | 421224000   | 0    | 5020
ps...        | 192...    | /producers  | pidIdx   | RANGE | id                 | /prod
ucers  | 0    | 5031    | 497872000   | 5031 | 5031
```

Error Messages:

```
No Indexes Found
```

list jndi-binding

List all JNDI bindings, active and configured. An active binding is one that is bound to the server’s JNDI context and is also listed in the cluster configuration. A configured binding is one that is listed in the cluster configuration, but may not be active on the servers.

Availability: Online. You must be connected in `gfsh` to a JMX Manager member to use this command.

Syntax:

```
list jndi-binding
```

Sample Output:

```
gfsh>list jndi-binding

Configured JNDI bindings:

Group Name | JNDI Name | JDBC Driver Class
---------- | --------- | ------------------------------------
cluster    | jndi1     | org.apache.derby.jdbc.EmbeddedDriver

Active JNDI bindings found on each member:

    Member      |        JNDI Name        | JDBC Driver Class
--------------- | ----------------------- | ------------------------------------------
----------
land-gifted-gun | java:UserTransaction    | org.apache.geode.internal.jta.UserTransact
ionImpl
land-gifted-gun | java:TransactionManager | org.apache.geode.internal.jta.TransactionM
anagerImpl
```

list lucene indexes

Display the list of Lucene indexes created for all members. The optional `--with-stats` qualifier shows activity on the indexes.

See also create lucene index, describe lucene index, destroy lucene index and search lucene.

Availability: Online. You must be connected in `gfsh` to a JMX Manager member to use this command.

Syntax:

```
list lucene indexes [--with-stats(=value)]
```

Parameters, list lucene indexes:

Name	Description	Default Value
--with-stats	Specifies whether statistics should also be displayed.	false if not specified, true if specified

Example Commands:

```
list lucene indexes
```

Sample Output:

```
gfsh>list lucene indexes --with-stats
Index Name | Region Path |     Indexed Fields     | Field Analy.. | Status   | Query Ex
ecutions | Updates | Commits | Documents
---------- | ----------- | ---------------------- | ------------- | -------- | --------
-------- | ------- | ------- | ---------
testIndex  | /testRegion | [__REGION_VALUE_FIELD] | {__REGION_V.. | Defined | NA
| NA       | NA       | NA

gfsh>list lucene indexes
 Index Name    | Region Path |                           Indexed Fields
| Field Analy.. | Status
------------- | ----------- | --------------------------------------------------------
---------- | ------------- | -----------
analyzerIndex | /Person     | [revenue, address, name, email]
| {revenue=St.. | Initialized
customerIndex | /Customer   | [symbol, revenue, SSN, name, email, address, __REGION_VA
LUE_FIELD] | {}            | Initialized
pageIndex     | /Page       | [id, title, content]
| {}            | Initialized
personIndex   | /Person     | [name, email, address, revenue]
| {}            | Initialized
```

list members

Display all or a subset of members.

Within the output, the membership coordinator is listed. `<vN>` identifies which view the member currently has; `N` will be zero or a positive integer. `<ec>` indicates which members are eligible to be a membership coordinator.

Availability: Online. You must be connected in `gfsh` to a JMX Manager member to use this command.

Syntax:

```
list members [--group=value]
```

Parameters, list members:

Name	Description
--group	Group name for which members will be displayed.

Example Commands:

```
list members
list members --group=Group1
```

Sample Output:

```
gfsh>list members
  Name         | Id
------------ | -------------------------------------
Coordinator: | 192.0.2.0(locator1:216:locator)<ec><v6>:33368
locator1     | 192.0.2.0(locator1:216:locator)<ec><v6>:33368
server1      | 192.0.2.0(server1:888)<v7>:10839
server2      | 192.0.2.0(server2:3260)<v8>:16721
```

list regions

Display regions of a member or members. If no parameter is specified, all regions in the cluster are listed.

Syntax:

```
list regions [--groups=value(,value)*] [--members=value(,value)*]
```

Parameters, list regions:

Name	Description
--groups	Group(s) of members for which regions will be displayed.
--members	Name or ID of the member(s) for which regions will be displayed.

Example Commands:

```
list regions
list regions --groups=G1
list regions --members=member1
```

Sample Output:

```
gfsh>list regions
List of regions
```

```
---------------
region1
region2
```

load-balance gateway-sender

Causes the specified gateway sender to close its current connections and reconnect to remote gateway receivers in a more balanced fashion.

Use this command to load balance connections between gateway senders to receivers. For example, when you add a new gateway receiver node at a remote site, execute this command so that the new gateway receiver can pick up connections from the specified gateway sender. Invoking this command redistributes a sender's connections more evenly among all the gateway receivers.

Note: This command has no effect on ping connections.

Availability: Online. You must be connected in `gfsh` to a JMX Manager member to use this command.

Syntax:

```
load-balance gateway-sender --id=value
```

Name	Description
--id	*Required.* ID of the Gateway Sender.

Table 1. Load-Balance Gateway-Sender Parameters

Example Commands:

```
load-balance gateway-sender --id=sender1-LN
```

Sample Output:

```
load-balance gateway-sender --id=ny

                 Member              | Result | Message
  --------------------------------- | ------ |----------------------------------------
---------------------------------
boglesbymac(ln-1:88651)<v2>:48277 | OK     | GatewaySender ny is rebalanced on member
boglesbymac(ln-1:88651)<v2>:48277
boglesbymac(ln-4:88681)<v5>:42784 | OK     | GatewaySender ny is rebalanced on member
boglesbymac(ln-4:88681)<v5>:42784
boglesbymac(ln-3:88672)<v4>:43675 | OK     | GatewaySender ny is rebalanced on member
boglesbymac(ln-3:88672)<v4>:43675
boglesbymac(ln-2:88662)<v3>:12796 | OK     | GatewaySender ny is rebalanced on member
boglesbymac(ln-2:88662)<v3>:12796
```

locate entry

Locate a region entry on a member.

locate entry

Locate a given entry on members using the specified key. This command is useful when using partitioned regions.

Availability: Online. You must be connected in `gfsh` to a JMX Manager member to use this command.

Syntax:

```
locate entry --key=value --region=value [--key-class=value]
[--value-class=value] [--recursive=value]
```

Name	Description	Default Value
--key	*Required.* String or JSON text from which to create a key. Examples include: “`James`”, “`100L`” and “`('id': 'l34s')`”.	
--region	*Required.* Region in which to locate values.	
--key-class	Fully qualified class name of the key’s type.	`java.lang.String`
--value-class	Fully qualified class name of the value’s type.	`java.lang.String`
--recursive	Whether to traverse regions and subregions recursively.	false

Table 1. Locate Entry Parameters

Example Commands:

```
locate entry --key=('id':'133abg124') --region=/region1
--key-class=data.ProfileKey --recursive=true;
```

Sample Output:

```
gfsh>locate entry --key=('123abc') --region=region2
Result          : true
Key Class       : java.lang.String
Key             : ('123abc')
Locations Found : 2


MemberName | MemberId
---------- | -------------------------------------
server1    | GemFireStymon(server1:3692)<v1>:13487
server2    | GemFireStymon(server2:2340)<v2>:11613
```

netstat

Report network information and statistics via the “netstat” operating system command.

Report important network usage information/statistics for the given member.

Availability: Online. You must be connected in `gfsh` to a JMX Manager member to use this command.

Syntax:

```
netstat [--members=value(,value)*] [--group=value] [--file=value]
[--with-lsof(=value)?]
```

Name	Description	Default Value
--members	Name or ID of the member(s) on which to run the `netstat` command.	
--group	Group of members on which to run the `netstat` command.	
--file	Text file to which output from the `netstat` command will be written. A "`.txt`" extension will be added if it is not already part of the specified name.	
--with-lsof	Specifies whether `lsof` (list open files) command output should also be displayed. Not applicable for Microsoft Windows hosts.	false

Table 1. Netstat Parameters

Example Commands:

```
netstat
netstat --members=server1
netstat --members=server1 --file=server1_netstat.txt
```

Sample Output:

```
gfsh>netstat
####################################
Host: GemFireTest
OS: Windows XP 5.1 x86
Member(s):
 server2, locator1, server1
####################################

Active Connections

  Proto  Local Address          Foreign Address        State
  TCP    GemFireTest:epmap    GemFireTest:0        LISTENING
  TCP    GemFireTest:microsoft-ds  GemFireTest:0        LISTENING
  TCP    GemFireTest:1034     GemFireTest:0        LISTENING
  TCP    GemFireTest:1069     GemFireTest:0        LISTENING
  TCP    GemFireTest:1099     GemFireTest:0        LISTENING
  TCP    GemFireTest:1134     GemFireTest:0        LISTENING
  TCP    GemFireTest:3389     GemFireTest:0        LISTENING
  TCP    GemFireTest:8080     GemFireTest:0        LISTENING
  TCP    GemFireTest:8081     GemFireTest:0        LISTENING
  TCP    GemFireTest:10334    GemFireTest:0        LISTENING
  TCP    GemFireTest:40404    GemFireTest:0        LISTENING
  TCP    GemFireTest:40405    GemFireTest:0        LISTENING
  TCP    GemFireTest:1025     GemFireTest:0        LISTENING
  TCP    GemFireTest:5152     GemFireTest:0        LISTENING
  TCP    GemFireTest:netbios-ssn  GemFireTest:0        LISTENING
  TCP    GemFireTest:1035     GemFireTest:0        LISTENING
  TCP    GemFireTest:1035     GemFireTest:1081     ESTABLISHED
  TCP    GemFireTest:1035     GemFireTest:1086     ESTABLISHED
  TCP    GemFireTest:1035     GemFireTest:1147     ESTABLISHED
  TCP    GemFireTest:1035     GemFireTest:1156     ESTABLISHED
  TCP    GemFireTest:1046     GemFireTest:1099     ESTABLISHED
  TCP    GemFireTest:1049     osdc-proxy-vip.vmware.com:http  CLOSE_WA
  TCP    GemFireTest:1050     osdc-proxy-vip.vmware.com:3128  CLOSE_WA
  TCP    GemFireTest:1071     GemFireTest:0        LISTENING
  TCP    GemFireTest:1071     GemFireTest:1077     ESTABLISHED
  TCP    GemFireTest:1071     GemFireTest:1150     ESTABLISHED
  TCP    GemFireTest:1071     GemFireTest:1157     ESTABLISHED
  TCP    GemFireTest:1077     GemFireTest:1071     ESTABLISHED
  TCP    GemFireTest:1078     GemFireTest:24400    ESTABLISHED
  TCP    GemFireTest:1081     GemFireTest:1035     ESTABLISHED
  TCP    GemFireTest:1086     GemFireTest:1035     ESTABLISHED
  TCP    GemFireTest:1099     GemFireTest:1046     ESTABLISHED
  TCP    GemFireTest:1136     GemFireTest:0        LISTENING
  TCP    GemFireTest:1136     GemFireTest:1143     ESTABLISHED
  TCP    GemFireTest:1136     GemFireTest:1151     ESTABLISHED
  TCP    GemFireTest:1136     GemFireTest:1201     ESTABLISHED
  TCP    GemFireTest:1141     GemFireTest:4247     ESTABLISHED
  TCP    GemFireTest:1142     GemFireTest:48640    ESTABLISHED
  TCP    GemFireTest:1143     GemFireTest:1136     ESTABLISHED
  TCP    GemFireTest:1147     GemFireTest:1035     ESTABLISHED
  TCP    GemFireTest:1150     GemFireTest:1071     ESTABLISHED
  TCP    GemFireTest:1151     GemFireTest:1136     ESTABLISHED
  TCP    GemFireTest:1156     GemFireTest:1035     ESTABLISHED
  TCP    GemFireTest:1157     GemFireTest:1071     ESTABLISHED
  TCP    GemFireTest:1201     GemFireTest:1136     ESTABLISHED
  TCP    GemFireTest:1349     GemFireTest:10334    TIME_WAIT
  TCP    GemFireTest:1350     GemFireTest:10334    TIME_WAIT
  TCP    GemFireTest:1351     GemFireTest:10334    TIME_WAIT
  TCP    GemFireTest:1352     GemFireTest:10334    TIME_WAIT
  TCP    GemFireTest:1353     GemFireTest:10334    TIME_WAIT
  TCP    GemFireTest:1354     GemFireTest:10334    TIME_WAIT
  TCP    GemFireTest:4247     GemFireTest:0        LISTENING
  TCP    GemFireTest:4247     GemFireTest:1141     ESTABLISHED
  TCP    GemFireTest:24400    GemFireTest:0        LISTENING
  TCP    GemFireTest:24400    GemFireTest:1078     ESTABLISHED
  TCP    GemFireTest:48640    GemFireTest:0        LISTENING
  TCP    GemFireTest:48640    GemFireTest:1142     ESTABLISHED
  UDP    GemFireTest:microsoft-ds  *:*
  UDP    GemFireTest:isakmp   *:*
  UDP    GemFireTest:4500     *:*
  UDP    GemFireTest:ntp      *:*
  UDP    GemFireTest:1900     *:*
  UDP    GemFireTest:ntp      *:*
  UDP    GemFireTest:netbios-ns  *:*
  UDP    GemFireTest:netbios-dgm  *:*
  UDP    GemFireTest:1900     *:*
  UDP    GemFireTest:32270    *:*
  UDP    GemFireTest:42838    *:*
  UDP    GemFireTest:47727    *:*
```

pause gateway-sender

Pause a gateway sender.

Pause the gateway sender on a member or members. See Pausing Gateway Senders for details on pausing gateway senders.

Availability: Online. You must be connected in `gfsh` to a JMX Manager member to use this command.

Syntax:

```
pause gateway-sender --id=value [--groups=value(,value)*]
[--members=value(,value)*]
```

Name	Description
--id	*Required.* ID of the gateway sender.
--groups	Group(s) of members on which to pause the gateway sender.
--members	Name or ID of the member(s) on which to pause the gateway sender.

Table 1. Pause Gateway-Sender Parameters

Example Commands:

```
pause gateway-sender --id=sender1
```

pdx rename

Renames PDX types in an offline disk store.

Any PDX types that are renamed will be listed in the output. If no renames are done or the disk-store is online, then this command will fail.

Availability: Offline.

Syntax:

```
pdx rename --old=value --new=value --disk-store=value --disk-dirs=value(,value)*
```

Name	Description
--old	*Required.* If a PDX type's fully qualified class name has a word that matches this text then it will be renamed. Words are delimited by '.' and '$'.
--new	*Required.* The text to replace the word that matched old.
--disk-store	*Required.* Name of the disk store to operate on.
--disk-dirs	*Required.* Directories which contain the disk store files.

Example Commands:

Change all packages that start with "com.gemstone" to "com.pivotal":

```
gfsh>pdx rename --old=com.gemstone --new=com.pivotal --disk-store=ds1 --disk-dirs=/dis
kDir1
```

Change a class named "MyClassName" to "YourClassName":

```
gfsh>pdx rename --old=MyClassName --new=YourClassName --disk-store=ds1 --disk-dirs=/di
skDir1
```

Change the FQCN “com.target.app1.OldClass” to “com.target.app2.NewClass”:

```
gfsh>pdx rename --old=com.target.app1.OldClass --new=com.target.app2.NewClass --disk-s
tore=ds1 --disk-dirs=/diskDir1
```

Sample Output:

```
gfsh>pdx rename --old=PortfolioPdx --new=StockPdx --disk-store=DiskStore1 --disk-dirs
=/DiskDir1
Successfully renamed pdx types:
  com.app.data.StockPdx: id=2
  com.app.data.StockPdx$Day.Monday
```

Error Messages:

If no types match, you may receive the following error message:

```
gfsh>pdx rename --old=gemstone --new=pivotal --disk-store=DiskStore1 --disk-dirs=/Disk
Dir1
Could not process command due to GemFire error. No Pdx types found to rename.
```

If the disk-store where the PDX types are stored is online, you will receive the following error message:

```
gfsh>pdx rename --old=StockPdx --new=PortfolioPdx --disk-store=DiskStore1 --disk-dirs
=/DiskDir1
Could not process command due to GemFire error. Error renaming pdx types :
GemFireCache[id = 484629896; isClosing = false; isShutDownAll = false;
closingGatewayHubsByShutdownAll = false; created = Wed Jan 07 10:29:45 PST 2015;
server = false; copyOnRead = false; lockLease = 120; lockTimeout = 60]: An open cache
already exists.
```

put

Add or update a region entry.

Availability: Online. You must be connected in `gfsh` to a JMX Manager member to use this command.

Syntax:

```
put --key=value --value=value --region=value [--key-class=value]
[--value-class=value] [--if-not-exists(=value)]
```

Name	Description	Default Value
--key	*Required.* String or JSON text from which to create the key. For example: “`James`”, “`100L`” and “`('id': 'l34s')`”.	
--value	*Required.* String or JSON text from which to create the value. For example: “`James`”, “`100L`” and “`('id': 'l34s')`”.	

Name	Description	Default Value
--region	*Required*. Region into which the entry will be put.	
--key-class	Fully qualified class name of the key's type.	`java.lang.String`
--value-class	Fully qualified class name of the value's type. The type may not be an internal PDX class.	`java.lang.String`
--value-class	Fully qualified class name of the value's type. The type may not be an internal PDX class.	`java.lang.String`
--if-not-exists	Skip the put operation when an entry with the same key already exists.	false

Example Commands:

```
put --key=('id':'133abg125') --value=('firstname':'James','lastname':'Gosling')
--region=/region1 --key-class=data.ProfileKey --value-class=data.ProfileDetails

put --key=('133abg124') --value=('Hello World!!') --region=/region2

put --key=('100F') --value=('2146547689879658564') --region=/region1/region12
--key-class=java.lang.Float --value-class=java.lang.Long
```

Sample Output:

```
gfsh>put --key=('123abc') --value=('Hello World!!') --region=region2
Result      : true
Key Class   : java.lang.String
Key         : ('123abc')
Value Class : java.lang.String
Old Value   : <NULL>
```

Error Messages:

```
"Region name is either empty or Null";
"Key is either empty or Null";
"Value is either empty or Null";
"Region <{0}> not found in any of the members";
"Region <{0}> Not Found";
"Key is not present in the region";
```

query

Run queries against VMware GemFire regions.

If a limit restricting the result size is not set in the query, then a default limit of the gfsh environment variable `APP_FETCH_SIZE`, as defined in Useful gfsh Shell Variables, will be applied.

Surround the OQL query with single quote marks.

Note: This command should not be executed from `gfsh` if the objects being queried contain cyclic references.

Availability: Online. You must be connected in `gfsh` to a JMX Manager member to use this command.

Syntax:

```
query --query=value [--step-name=value] [--file=path/to/results/file]
```

Name	Description
--query	*Required.* The OQL string.
--file	When specified, all query results are written to the specified file. An error is issued if the file already exists.

Table 1. Query Parameters

Sample Output:

```
gfsh>query --query='SELECT * FROM /region2'

Result     : true
startCount : 0
endCount   : 20
Rows       : 1

Result
-----------------
('Hello World!!')

NEXT_STEP_NAME : END
```

rebalance

Rebalance partitioned regions.

The default is for all partitioned regions to be rebalanced.

Availability: Online. You must be connected in `gfsh` to a JMX Manager member to use this command.

Syntax:

```
rebalance [--include-region=value(,value)*]
[--exclude-region=value(,value)*] [--time-out=value] [--simulate(=value)?]
```

Name	Description	Default Value
--include-region	Partitioned Region paths to be included for rebalance operation. Includes take precedence over excludes.	
--exclude-region	Partitioned Region paths to be excluded for rebalance operation.	
--time-out	Time to wait (in seconds) before GFSH returns to a prompt while rebalancing continues in the background.	-1 (wait for rebalancing to complete
--simulate	Whether to only simulate rebalancing. The –time-out parameter is not available when simulating.	false

Table 1. Rebalance Parameters

Example Commands:

```
rebalance --include-region=/region3 --simulate=true
```

Sample Output:

```
rebalance
1. server1  host1(3467):12435:12423
Row  Rebalanced Stats               | Value
---  ----------------               | -----
1    TotalBucketCreateBytes         | 0
2    TotalBucketCreateTime          | 0
3    TotalBucketCreatesCompleted    | 0
4    TotalBucketTransferBytes       | 0
5    TotalBucketTransferTime        | 0
6    TotalBucketTransfersCompleted  | 0
7    TotalPrimaryTransferTime       | 0
8    TotalPrimaryTransfersCompleted | 0
9    TotalTime                      | 56

Rebalance complete on host1(3467):12435:12423.
```

remove

Remove an entry from a region.

Availability: Online. You must be connected in `gfsh` to a JMX Manager member to use this command.

Syntax:

```
remove --region=value [--key=value] [--all(=value)?] [--key-class=value]
```

Name	Description	Default Value
--key	String or JSON text that will be used to create a key to retrieve a value .	
--key-class	Fully qualified class name of the key's type.	key constraint for the current region or String
--region	*Required.* Region from which to remove the entry.	
--all	A boolean value that, when true, clears the region by removing all entries. This option is not available for partitioned regions.	false

Example Commands:

```
gfsh>remove --region=/region1 --key=('id': '133abg134')
gfsh>remove --region=/region1 --key=('id': '133abg134') --key-class=data.ProfileKey
gfsh>remove --region=/region1 --all=true
```

Error Messages:

```
"Region name is either empty or Null"

"Key is either empty or Null"

"Value is either empty or Null"

"Region <{0}> not found in any of the members"

"Region <{0}> Not Found"

"Key is not present in the region"

"Option --all is not supported on partitioned region"
```

resume

Modify an existing VMware GemFire resource.

- **resume async-event-queue-dispatcher**

  Resume dispatching of events from the queue to the listener(s) for a specified asynchronous event queue which is currently in a paused state.

- **resume gateway-sender**

  Resume specified gateway sender that is currently in a paused state.

resume async-event-queue-dispatcher

Resume dispatching of events on a specified asynchronous event queue which is currently in a paused state.

Availability: Online. You must be connected in `gfsh` to a JMX Manager member to use this command.

Syntax:

```
resume async-event-queue-dispatcher --id=value [--groups=value(,value)*]
[--members=value(,value)*]
```

Name	Description
--id	*Required*. ID of the Asynchronous Event Queue.
--groups	Group(s) of members on which to resume event dispatching for this queue.
--members	Name/Id of the member(s) on which to resume event dispatching for this queue.

Example Commands:

```
resume async-event-queue-dispatcher --id=AEQ1 --groups=Group1
```

resume gateway-sender

Resume a gateway sender which is currently in a paused sate.

Availability: Online. You must be connected in `gfsh` to a JMX Manager member to use this command.

Syntax:

```
resume gateway-sender --id=value [--groups=value(,value)*]
[--members=value(,value)*]
```

Name	Description
--id	*Required.* ID of the Gateway Sender.
--groups	Group(s) of members on which to resume the Gateway Sender.
--members	Name/Id of the member(s) on which to resume the Gateway Sender.

Example Commands:

```
resume gateway-sender --id=sender1-LN --groups=LN-Group1
```

revoke missing-disk-store

Instruct the member(s) of a cluster to stop waiting for a disk store to be available.

Only revoke a disk store if its files are lost as it will no longer be recoverable after revoking is initiated. Use the "show missing-disk-store" command to get descriptions of missing disk stores.

Availability: Online. You must be connected in `gfsh` to a JMX Manager member to use this command.

Syntax:

```
revoke missing-disk-store --id=value
```

Name	Description
--id	*Required.* ID of the missing disk store to be revoked.

Table 1. Revoke Missing-Disk-Store Parameters

Example Commands:

```
revoke missing-disk-store --id=60399215-532b-406f-b81f-9b5bd8d1b55a
```

Sample Output:

```
gfsh>revoke missing-disk-store --id=60399215-532b-406f-b81f-9b5bd8d1b55a
Missing disk store successfully revoked
```

Error Messages:

Example of `revoke missing-disk-store` when the disk store cannot be found:

```
gfsh> revoke missing-disk-store --id=60399215-532b-406f-b81f
```

```
Unable to find missing disk store to revoke
```

run

Execute a set of GFSH commands.

Commands that normally prompt for additional input will instead use default values.

Availability: Online or offline.

Note: Some commands specified in the file require online status.

Syntax:

```
run --file=value [--quiet(=value)?] [--continue-on-error(=value)?]
```

Table 1. Parameters

Name	Description	Default Value
--file	*Required.* Path of the file scripted with commands that `gfsh` recognizes. Path should be relative or absolute.	
--quiet	Specifies whether to show command output. **Note:** All commands in a script are run non-interactively when the `run` command is used. This option does not change that functionality.	false
--continue-on-error	Specifies whether further execution of the script should continue if there is an error while executing one of the commands fails.	false

Table 1. Run Parameters

Example Commands:

```
run --file=create-regions.gfsh --quiet=true

(2) From command line:
prompt> /home/user1/gemfire70/bin/gfsh run ./create-regions.gfsh --quiet=true
prompt> /home/user1/gemfire70/bin/gfsh run ./create-regions.gfsh
--continue-on-error=true
```

Sample Output:

```
gfsh>run --file=create-regions.gfsh
1. Executing - create region --name=region4 --type=REPLICATE

Member  | Status
------- | --------------------------------------
server2 | Region "/region4" created on "server2"
server1 | Region "/region4" created on "server1"

2. Executing - create region --name=region1/subregion1 --type=LOCAL
```

```
Parent region for "region1/subregion1" doesn't exist.
```

search lucene

search lucene

Search a Lucene index

See also create lucene index, describe lucene index, destroy lucene index and list lucene indexes.

Availability: Online.

Syntax:

```
search lucene --name=value --region=value --queryString=value --defaultField=value
    [--limit=value] [--keys-only=value]
```

Parameters, search lucene:

Name	Description	Default Value
--name	*Required*. Name of the Lucene index to search.	
--region	*Required*. Name/Path of the region where the Lucene index exists.	
--queryString	*Required*. Query string to search the Lucene index. Use `__REGION_VALUE_FIELD` as the field name within the query string when the field is a primitive value. Surround a string with double quote marks to do an exact match of the string.	
--defaultField	*Required*. Default field to search in. `__REGION_VALUE_FIELD` identifies the field as a primitive value.	
--limit	Number of search results needed.	If the parameter is not specified: `100`
--keys-only	Return only keys of search results.	If the parameter is not specified: `false`

Example Commands:

```
gfsh> search lucene --name=testIndex --region=/testRegion --queryString=value1
   --defaultField=__REGION_VALUE_FIELD


gfsh> search lucene --name=indexOfStrings --region=/stringTestRegion
      --queryString='__REGION_VALUE_FIELD:"my exact string"'
      --defaultField=__REGION_VALUE_FIELD
```

Sample Output:

```
gfsh>search lucene --name=testIndex --region=/testRegion --queryString=value*
   --defaultField=__REGION_VALUE_FIELD
key | value  | score
--- | ------ | -----
3   | value3 | 1
2   | value2 | 1
1   | value1 | 1
```

```
gfsh>search lucene --region=/Person --name=analyzerIndex
   --defaultField=addr --queryString="97763"
 key   |                         value                                    | score
------ | ---------------------------------------------------------------- | --------
key763 | Person{name='Kris Cat', addr='7 Ash St, Portland_OR_97763', emai.. | 1.669657
```

set variable

Set variables in the gfsh environment.

Set gfsh variables that can be used by commands.

You can use the gfsh echo command to view the value of a variable. For example, to see a list of all environment variables and their current values, use the following command:

```
gfsh>echo --string=$*
```

See Useful gfsh Shell Variables for a description of preset environment variables.

Availability: Online or offline.

Syntax:

```
set variable --name=value --value=value
```

Name	Description
--name	*Required*. The name to be assigned for the variable. The name must only be composed of letters, digits, and the underscore character (_); the name may not start with a digit.
--value	*Required*. Value that the variable will be set to.

Example Commands:

```
set variable --name=FOO --value="foo"
set variable --name=BAR --value="bar"
```

Sample Output:

```
gfsh>set variable --name=BAR --value="bar"
Value for variable BAR is now: "bar".
```

sh

Execute operating system commands.

Executes operating system (OS) commands. Use "`&`" to return to the `gfsh` prompt immediately.

Note: Commands that pass output to another shell command are not supported.

Syntax:

```
sh command [--use-console(=value)?]
```

Name	Description	Default Value
--use-console	Set this parameter on UNIX systems for applications which need a handle to the console. Adds "`</dev/tty >/dev/tty`" to the specified command.	false

Table 1. Sh Parameters

Example Commands:

```
gfsh>sh ls -al
total 80
drwxrwxr-x. 10 username username  4096 Sep  3 15:10 .
drwxrwxr-x.  4 username username  4096 Sep  3 14:58 ..
drwx------.  2 username username  4096 Sep  3 15:09 bin
drwx------.  2 username username  4096 Sep  3 15:09 defaultConfigs
drwx------.  3 username username  4096 Sep  3 15:09 docs
drwx------.  2 username username  4096 Sep  3 15:09 dtd
-rwx------.  1 username username 31830 Sep  3 15:09 EULA.txt
drwx------.  2 username username  4096 Sep  3 15:09 lib
drwx------.  6 username username  4096 Sep  3 15:09 SampleCode
drwx------.  4 username username  4096 Sep  3 15:09 templates
drwx------.  5 username username  4096 Sep  3 15:09 tools
```

show

Display deadlocks, logs, metrics and missing disk-stores.

- **show dead-locks**

  Display any deadlocks in the cluster.

- **show log**

  Display the log for a member.

- **show metrics**

  Display or export metrics for the entire cluster, a member, or a region.

- **show missing-disk-stores**

  Display a summary of the disk stores that are currently missing from the cluster.

- **show subscription-queue-size**

  Shows the number of events in the subscription queue.

show dead-locks

Display any deadlocks in the cluster.

Availability: Online. You must be connected in `gfsh` to a JMX Manager member to use this command.

Syntax:

```
show dead-locks --file=value
```

Name	Description
--file	*Required.* Name of the file to which dependencies between members will be written.

Table 1. Show Dead-Locks Parameters

Example Commands:

```
show dead-locks --file=deadlocks.txt
```

Sample Output:

```
gfsh>show dead-locks --file=deadlocks.txt
No dead lock detected.

Please view the dependencies between the members in file : deadlocks.txt
```

show log

Display the log for a member.

Availability: Online. You must be connected in `gfsh` to a JMX Manager member to use this command.

Syntax:

```
show log --member=value [--lines=value]
```

Name	Description	Default Value
--member	*Required.* Name/ID of the member whose log file will be displayed.	
--lines	Number of lines from the log file to display. The maximum is 100.	0

Example Commands:

```
show log --member=locator1 --lines=5
```

Sample Output:

```
gfsh>show log --member=locator1 --lines=5
SystemLog:
[info 2012/09/25 14:04:51.340 PDT locator1 <RMI TCP Connection(14)-192.0.2.0> tid=0x5
7] (tid=12 msgId=4) Parent region for "region1/subregion1" doesnt exi
st.

[info 2012/09/25 14:04:51.372 PDT locator1 <RMI TCP Connection(14)-192.0.2.0> tid=0x5
7] (tid=12 msgId=5) Error occurred while executing "create region --n
ame=region1/subregion1 --type=LOCAL".

[info 2012/09/25 15:14:34.314 PDT locator1 <RMI TCP Connection(159)-192.0.2.0> tid=0x6
8] (tid=13 msgId=6) Error occurred while executing "show log --membe
r=server1 --lines=5".
```

show metrics

Display or export metrics for the entire cluster, a member, or a region.

When no command line arguments are given, metrics under the categories of cluster, cache, diskstore, and query are displayed.

Availability: Online. You must be connected in `gfsh` to a JMX Manager member to use this command.

Syntax:

```
show metrics [--member=value] [--region=value] [--file=value]
[--port=value] [--categories=value(,value)*]
```

Table 3. Show Metrics Parameters

Name	Description
--member	Name/ID of the member whose metrics will be displayed/exported.
--region	Name/Path of the region whose metrics will be displayed/exported.
--file	Name of the file to which metrics will be written.
--port	Port number of the Cache Server whose metrics are to be displayed/exported. This can only be used along with the `--member` parameter.
--categories	Categories available based upon the parameters specified (listed above) are: • **region specified**: cluster, region, partition, diskstore, callback, eviction • **member specified**: member, jvm, region, serialization, communication, function, transaction, diskstore, lock, eviction, distribution, offheap • **member and region specified**: region, partition, diskstore, callback, eviction

Table 3. Show Metrics Parameters

Example Commands:

```
// Metrics for the entire system
show metrics
// Metrics for a region:
show metrics --region=region1
// Metrics for a given member
```

```
show metrics --member=server1
// Metrics for a region on a member
show metrics --region=region1 --member=server1
//  Metrics for a member and the cacheserver it hosts
// NOTE: port option only work when used with --member option
show metrics --member=server1 --port=10334
// Export metrics for the entire system
show metrics --file=data/stats/system-stats.csv
```

Sample Output:

```
gfsh>show metrics

Cluster-wide Metrics

  Type    |        Metric         | Value
--------- | --------------------- | -----
cluster   | totalHeapSize         | 123
cache     | totalRegionEntryCount | 0
          | totalRegionCount      | 0
          | totalMissCount        | 0
          | totalHitCount         | 0
diskstore | totalBytesOnDisk      | 0
          | diskReadsRate         | 0
          | diskWritesRate        | 0
          | flushTimeAvgLatency   | 0
          | totalBackupInProgress | 0
query     | queryRequestRate      | 0
```

show missing-disk-stores

Display a summary of the disk stores that are currently missing from the cluster.

Availability: Online. You must be connected in `gfsh` to a JMX Manager member to use this command.

Syntax:

```
show missing-disk-stores
```

Example Commands:

```
show missing-disk-stores
```

Sample Output:

```
gfsh> show missing-disk-stores

           Disk Store ID             |   Host    |               Directory
------------------------------------ | --------- | -----------------------------------
--
60399215-532b-406f-b81f-9b5bd8d1b55a | excalibur | /usr/local/gemfire/deploy/disk_stor
e1
```

show subscription-queue-size

Shows the number of events in the subscription queue.

If a CQ name is provided, it counts the number of events in the subscription queue for the specified CQ.

Availability: Online. You must be connected in `gfsh` to a JMX Manager member to use this command.

Syntax:

```
show subscription-queue-size --durable-client-Id=value
 [--members=value(,value)*] [--groups=value(,value)*]
```

Name	Description
--durable-client-id	*Required.* The ID used to identify the durable client.
--durable-cq-name	The name that identifies the CQ.
--members	Name/Id of the member(s) for which subscription queue events are to be counted.
--groups	Group(s) of members for which subscription queue events are to be counted.

Table 4. Show Subscription-Queue-Size Parameters

Example Commands:

```
show subscription-queue-size --durable-client-id=client1
```

Sample Output:

```
gfsh>show subscription-queue-size --durable-client-Id=client1
member  | subcription-queue-size for durable-client : "client1".
------- | ------------------------------------------------------
server3 | 1
server4 | 0
```

Error Messages:

```
gfsh>show subscription-queue-size --durable-client-Id=client1

No client found with client-id : client1
Occurred on members
1.server4
2.server1
3.server3
```

shutdown

Stop all members.

Asks all the members that have a cache to close the cache and disconnect from the system. If the `--include-locators` parameter is specified, the command shuts down any running locators one by one. The timeout parameter allows you to specify that the system should be shutdown forcibly after the time has exceeded.

Availability: Online. You must be connected in `gfsh` to a JMX Manager member to use this command.

Syntax:

```
shutdown [--time-out=value] [--include-locators=value]
```

Name	Description	Default Value
--time-out	Time to wait (in seconds) for a graceful shutdown. Should be at least 10 seconds. The default value if not specified is 10.	10
--include-locators	To shutdown locators, specify this option as true.	false

Table 1. Shutdown Parameters

Example Commands:

```
shutdown
shutdown --time-out=15
shutdown --include-locators=true
```

Sample Output:

```
gfsh>shutdown
"As a lot of data in memory will be lost, including possibly events in
queues, do you really want to shutdown the entire distributed system? (Y/n):"
Y
Shutdown is triggered
```

sleep

Delay `gfsh` command execution.

Delay for a specified amount of time in seconds - floating point values are allowed.

Availability: Online of offline.

Syntax:

```
sleep [--time=value]
```

Name	Description	Default Value
--time	Number of seconds to sleep.	3

Table 1. Sleep Parameters

Example Commands:

```
sleep
sleep --time=60
```

Sample Output:

```
gfsh>sleep --time=60

gfsh>
```

start

Start servers, locators, gateway senders and gateway receivers, and monitoring tools.

- **start gateway-receiver**

  Start the gateway receiver on a given member or group of members.

- **start gateway-sender**

  Start the gateway sender on a member or members.

- **start jconsole**

  Start the JDK JConsole monitoring application in a separate process.

- **start jvisualvm**

  Start the JDK's Java VisualVM monitoring application in a separate process.

- **start locator**

  Start a locator.

- **start pulse**

  Launch the VMware GemFire Pulse monitoring dashboard tool in the user's default system browser and navigates the user to the landing page (login page).

- **start server**

  Start a VMware GemFire cache server process.

- **start vsd**

  Launch GemFire Visual Statistics Display (VSD) in a separate process.

start gateway-receiver

Start the gateway receiver on a given member or group of members.

Note that you can only have one gateway receiver on each member, and unlike a gateway sender, you do not need to specify an identifier for the gateway receiver.

Availability: Online. You must be connected in `gfsh` to a JMX Manager member to use this command.

Syntax:

```
start gateway-receiver [--groups=value(,value)*] [--members=value(,value)*]
```

Parameters, start gateway-receiver

Name	Description
--members	Name or ID of the member(s) on which to start the Gateway Receiver.
--groups	Group(s) of members on which to start the Gateway Receiver.

Example Commands:

```
start gateway-receiver
start gateway-receiver --members=member1
```

Sample Output:

```
gfsh>start gateway-receiver
      Member                 | Result | Message
---------------------------- | -------| -----------------------------------------------
------------------------
pc13(2266)<v6>:56852         | OK     | GatewayReceiver is started on member pc13(2266)
<v6>:56852
pc13(Manager:2242)<v5>:57631| Error  | GatewayReceiver is not available on member pc13
(Manager:2242)<v5>:57631
pc13(2275)<v7>:47480         | OK     | GatewayReceiver is started on member pc13(2275)
<v7>:47480
pc13(2293)<v8>:55472         | OK     | GatewayReceiver is started on member pc13(2293)
<v8>:55472

gfsh>start gateway-receiver --members=pc13(2266)<v14>:36579
GatewayReceiver is started on member pc13(2266)<v14>:36579

gfsh>start gateway-receiver --group=RG1
         Member      | Result | Message
-------------------- | -------| -------------------------------------------------------
----
pc13(2275)<v23>:27484| OK     | GatewayReceiver is started on member pc13(2275)<v23>:2
7484
pc13(2293)<v24>:55810| OK     | GatewayReceiver is started on member pc13(2293)<v24>:5
5810
pc13(2266)<v22>:4522 | OK     | GatewayReceiver is started on member pc13(2266)<v22>:4
522
```

start gateway-sender

Start the gateway sender on a member or members.

For information on how to configure a gateway sender, see Configure Gateway Senders.

Note: By default, gateway senders are configured to start automatically. Manual restart introduces a risk of data loss; it is not intended for production systems.

Availability: Online. You must be connected in `gfsh` to a JMX Manager member to use this command.

Syntax:

```
start gateway-sender --id=value [--groups=value(,value)*] [--members=value(,value)*]
```

Parameters, start gateway-sender

Name	Description
--id	*Required.* ID of the GatewaySender.
--groups	Group(s) of members on which to start the Gateway Sender.
--members	Member(s) on which to start the Gateway Sender

Example Commands:

```
start gateway-sender --id=sender1-NY
start gateway-sender --id=sender1-NY --members=server1
start gateway-sender --id=sender1-NY --groups=MemberGroup1,MemberGroup2
```

Sample Output:

```
gfsh>start gateway-sender --id=ln
 Member                         |  Result |                   Message
------------------------------| ------- | --------------------------------------------
-----------------------------
pc13(30614)<v6>:63670          | OK      | GatewaySender ln is started on member pc13(3
0614)<v6>:63670
pc13(30621)<v7>:36015          | OK      | GatewaySender ln is started on member pc13(3
0621)<v7>:36015
pc13(30633)<v8>:13633          | OK      | GatewaySender ln is started on member pc13(3
0633)<v8>:13633
pc13(Manager:30588)<v5>:42792 | Error   | GatewaySender ln is not available on member
pc13(Manager:30588)<v5>:42792

gfsh>start gateway-sender --id=ln --members=pc13(30614)<v14>:44519
GatewaySender ln is started on member pc13(30614)<v14>:44519
gfsh>start gateway-sender --id=ln --groups=SenderGroup1
Member                 | Result| Message
---------------------- | ------| -----------------------------------------------------
-------
pc13(30614)<v18>:15201 | OK    | GatewaySender ln is started on member pc13(30614)<v18
>:15201
pc13(30621)<v19>:61437 | OK    | GatewaySender ln is started on member pc13(30621)<v19
>:61437
pc13(30633)<v20>:22567 | OK    | GatewaySender ln is started on member pc13(30633)<v20
>:22567
```

start jconsole

Start the JDK JConsole monitoring application in a separate process.

JConsole automatically connects to a running JMX Manager node if one is available.

Note that you must have a JDK installed (not just a JRE) and the correct PATH and JAVA_HOME environment variables set.

See Browsing VMware GemFire MBeans through JConsole for an example of using JConsole with the VMware GemFire management and monitoring system.

Availability: Online or offline.

Syntax:

```
start jconsole [--interval=<seconds>] [--notile] [--version]
[--J<jconsole JVM options>]
```

Parameters, start jconsole

Name	Description	Default Value
--interval	Set the update interval to n seconds (default is 4 seconds). (Equivalent to JConsole's `-interval=n`)	4
--notile	Whether to initially tile windows for two or more connections. This parameter is passed as `-notile` to JConsole.	false
--pluginpath	Directories or JAR files which are searched for JConsole plugins. The path should contain a provider-configuration file named:`META-INF/services/com.sun.tools.jconsole.JConsolePlugin` containing one line for each plugin specifying the fully qualified class name of the class implementing the `com.sun.tools.jconsole.JConsolePlugin` class.	
--version	Display the JConsole version information. This parameter is passed as `-version` to JConsole.	false
--J	Arguments passed to the JVM on which JConsole runs	

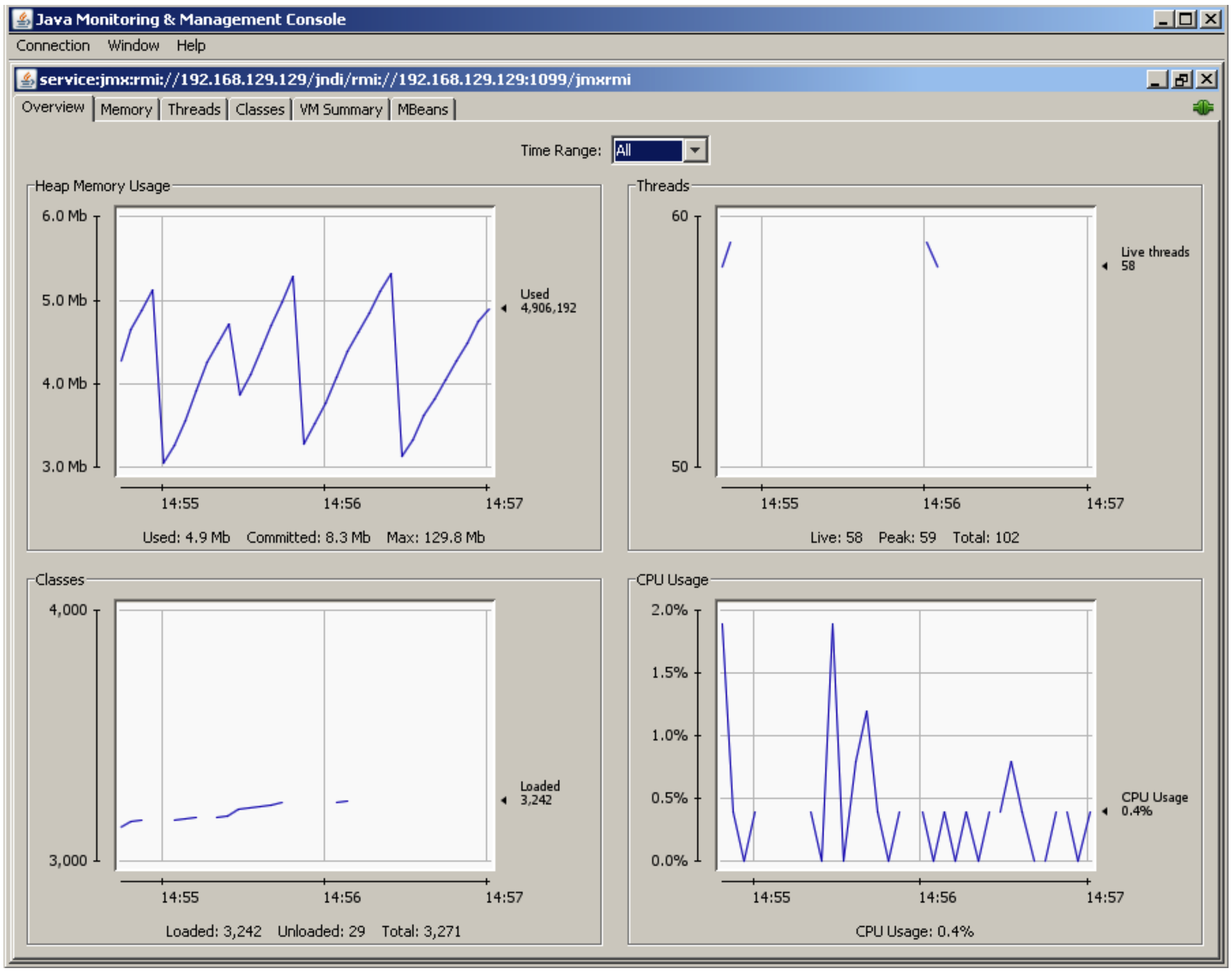
Example Commands:

```
gfsh>start jconsole --interval=8 --notile;
Running JDK JConsole

gfsh>start jconsole --version;
JConsole version "1.8.0_31-b01-1"
Java(TM) SE Runtime Environment (build 1.8.0_31-b01-1-11)
Java HotSpot(TM) 64-Bit Server VM (build 20.6-b01-11, mixed mode)
```

Sample Output:

```
gfsh>start jconsole
Running JDK JConsole
```

The JConsole application appears and auto-connects to a JMX Manager node if one is available:


Java Monitoring & Management Console
Connection Window Help
service:jmx:rmi://192.168.129.129/jndi/rmi://192.168.129.129:1099/jmxrmi
Overview Memory Threads Classes VM Summary MBeans
Time Range: All
Heap Memory Usage
Used 4,906,192
Used: 4.9 Mb Committed: 8.3 Mb Max: 129.8 Mb
Threads
Live threads 58
Live: 58 Peak: 59 Total: 102
Classes
Loaded 3,242
Loaded: 3,242 Unloaded: 29 Total: 3,271
CPU Usage
CPU Usage 0.4%
CPU Usage: 0.4%

Error Messages:

```
An error occurred while launching JConsole = %1$s

Connecting by the VMware GemFire member's name or ID is not currently supported.
Please specify the member as '<hostname|IP>[PORT].

An IO error occurred while launching JConsole.
Please ensure that JAVA_HOME is set to the JDK installation
or the JDK bin directory is in the system PATH.

JConsole could not be found.\nPlease ensure that JAVA_HOME is set to the
JDK installation or the JDK bin directory is in the system PATH.
```

start jvisualvm

Start the JDK's Java VisualVM monitoring application in a separate process.

Availability: Online or offline.

Syntax:

```
start jvisualvm [--J=value(,value)*]
```

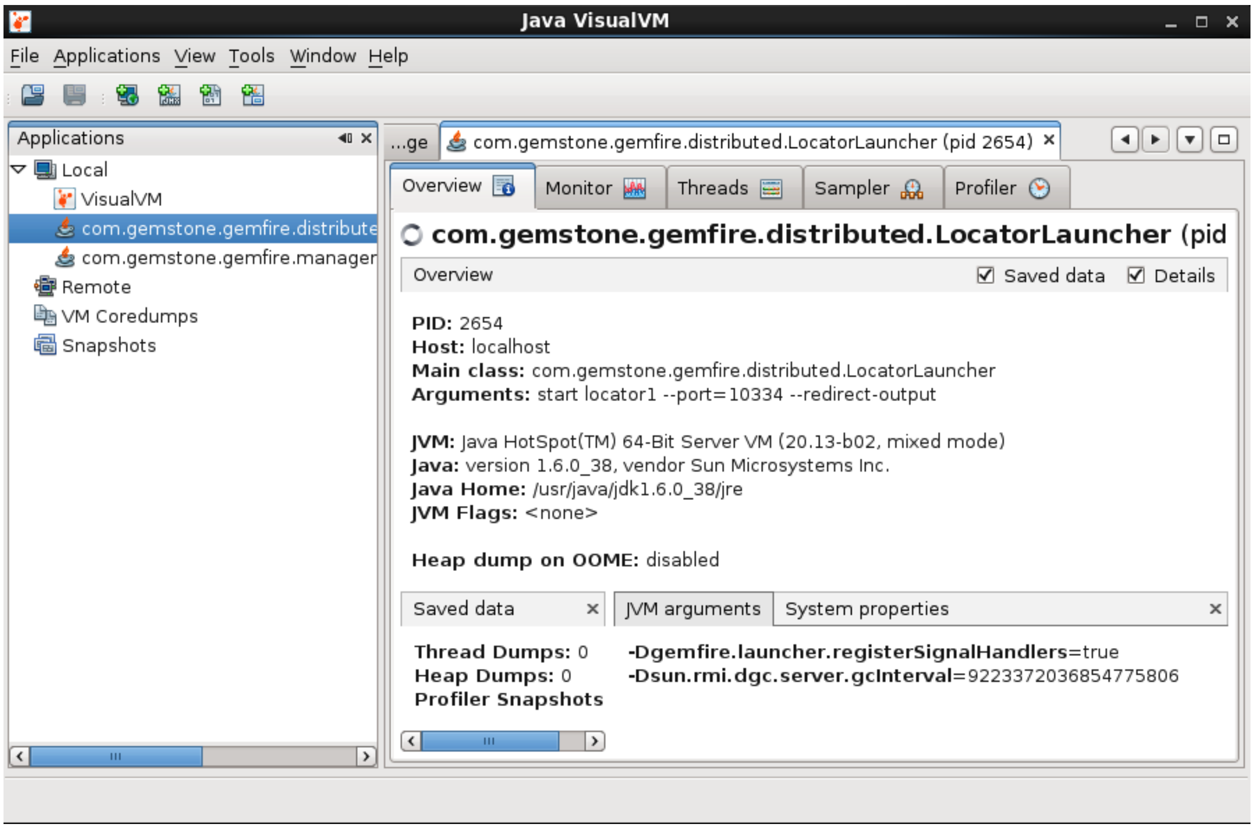
Parameters, start jvisualvm

Name	Description
--J	VM-option passed to the spawned CacheServer VM. For example: `-J-Dfoo.bar=true` for setting foo.bar to ‘true’.

Example Commands:

```
start jvisualvm
```

Sample Output:



com.gemstone.gemfire.distributed.LocatorLauncher (pid

Overview

PID: 2654
Host: localhost
Main class: com.gemstone.gemfire.distributed.LocatorLauncher
Arguments: start locator1 --port=10334 --redirect-output

JVM: Java HotSpot(TM) 64-Bit Server VM (20.13-b02, mixed mode)
Java: version 1.6.0_38, vendor Sun Microsystems Inc.
Java Home: /usr/java/jdk1.6.0_38/jre
JVM Flags: <none>

Heap dump on OOME: disabled

Thread Dumps: 0
Heap Dumps: 0
Profiler Snapshots

-Dgemfire.launcher.registerSignalHandlers=true
-Dsun.rmi.dgc.server.gcInterval=9223372036854775806

start locator

Start a locator.

The command creates a subdirectory and log file named after the locator. If the locator detects that no other JMX Manager exists, then the locator will automatically start an embedded JMX Manager and connect the current `gfsh` session to the JMX Manager.

Note: You must have `JAVA_HOME` set before starting gfsh to use this command.

In addition, if gfsh is not already connected to a JMX Manager, the gfsh console will automatically connect to the new embedded JMX Manager started by the new locator.

Note: When --max-heap is specified, gfsh enables the CMS garbage collector and sets `-XX:CMSInitiatingOccupancyFraction` to `60`. If you do not want gfsh to set these GC properties, then use the `-Xms` JVM option instead. See Controlling Heap Use with the Resource Manager for more information.

Note: The additional GC parameters introduced by the `--max-heap` option are not compatible with the G1 garbage collector.

Availability: Online or offline.

Syntax:

```
start locator --name=value [--bind-address=value] [--force(=value)]
 [--groups=value(,value)*] [--hostname-for-clients=value]
 [--locators=value] [--log-level=value] [--mcast-address=value] [--mcast-port=value]
[--port=value] [--dir=value]
 [--properties-file=value] [--security-properties-file=value] [--initial-heap=value]
[--max-heap=value]
 [--connect(=value)] [--enable-cluster-configuration(=value)] [--load-cluster-configur
ation-from-dir(=value)]
 [--cluster-config-dir=value]
 [--http-service-port=value] [--http-service-bind-address=value]
 [--J=value(,value)*]
```

Parameters, start locator

Name	Description	Default Value
--name	Name to be used for this VMware GemFire locator service. If not specified, gfsh generates a random name.	
--bind-address	IP address on which the locator will be bound.	bind to all addresses
--force	Whether to allow the PID file from a previous locator run to be overwritten.	false
--groups	Group(s) the locator will be a part of.	
--hostname-for-clients	Host name or IP address that will be sent to clients so they can connect to this locator.	uses `bind-address`
--locators	List of locators used by this locator to join the appropriate VMware GemFire cluster.	
--log-level	Level of output logged to the locator log file. Possible values for log-level include: ALL, TRACE, DEBUG, INFO, WARN, ERROR, FATAL, OFF.	
--mcast-address	IP address or hostname used to bind the UPD socket for multi-cast networking so the locator can locate other members in the VMware GemFire cluster. If mcast-port is zero, then mcast-address is ignored.	
--mcast-port	Port used for multi-cast networking so the locator can locate other members of the VMware GemFire cluster. A zero value deactivates mcast.	
--port	Port the locator will listen on.	10334
--dir	Directory in which the Locator will be started and run.	`.<locator-member-name>`
--properties-file	Specify the `gemfire.properties` file for configuring the locator's cluster. The file's path should be absolute or relative to gfsh's working directory.	
--security-properties-file	The `gfsecurity.properties` file for configuring the Locator's security configuration in the cluster. The file's path can be absolute or relative to gfsh's working directory.	
--initial-heap	Size has the same format as the `-Xmx`/`-Xms` JVM options.	

Name	Description	Default Value
--max-heap	Size has the same format as the `-Xmx`/`-Xms` JVM options. **Note:** The additional GC parameters introduced by the `-max-heap` option are not compatible with the usage of the G1 garbage collector.	
--connect	When connect is set to false, gfsh does not automatically connect to the locator which is started using this command.	true
--enable-cluster-configuration	Enables cluster configuration behavior where locators maintain configurations for all members of the cluster. See Overview of the Cluster Configuration Service.	true
--load-cluster-configuration-from-dir	**Deprecated. Use `gfsh import cluster-configuration` for this functionality.** Loads the cluster configuration from the `shared-config` directory. (When set to false, the configuration is loaded from the disk store of the internal, persistent region used by the locator to persist the configuration.)	false
--cluster-config-dir	Directory used by the cluster configuration service to store the cluster configuration on the filesystem	cluster-config
--http-service-port	Specifies the HTTP service port.	7070
--http-service-bind-address	Specifies the IP address to which the HTTP service will be bound.	the local host machine's address
--J	Argument passed to the JVM on which the Locator will run. For example, specifying `--J=-Dfoo.bar=true` sets property "foo.bar" to "true". **Note:** If the argument you are passing contains spaces or commas, enclose the option in single quotes. For example: `start locator --name=locator1 --port=9009 --mcast-port=0 --J='-Dgemfire.remote-locators=192.0.2.0[9009],192.0.2.1[9009]'`	none

Example Commands:

```
start locator --name=locator1
```

start pulse

Launch the VMware GemFire Pulse monitoring dashboard tool in the user's default system browser and navigate the user to the landing page (login page).

For more information on VMware GemFire Pulse, see VMware GemFire Pulse.

Availability: Online or offline.

Syntax:

```
start pulse [--url=value]
```

Parameters, start pulse

Name	Description	Default
--url	URL of the Pulse Web application	`http://localhost:7070/pulse`

Example Commands:

```
start pulse
start pulse --url=http://gemfire.example.com:7070/pulse
```

Sample Output: See VMware GemFire Pulse for examples of Pulse.

start server

Start a VMware GemFire cache server process.

Note: When --max-heap is specified, gfsh enables the CMS garbage collector and sets `-XX:CMSInitiatingOccupancyFraction` to `60`. If you do not want gfsh to set these GC properties, then use the `-Xms` JVM option instead. See Controlling Heap Use with the Resource Manager for more information.

Note: The additional GC parameters introduced by the `--max-heap` option are not compatible with the G1 garbage collector.

Availability: Online or offline.

Syntax:

```
start server --name=value [--assign-buckets(=value)] [--bind-address=value]
    [--cache-xml-file=value] [--classpath=value] [--disable-default-server(=value)]
    [--disable-exit-when-out-of-memory(=value)] [--enable-time-statistics(=value)]
    [--force(=value)] [--include-system-classpath(=value)] [--properties-file=value]
    [--security-properties-file=value] [--groups=value(,value)*]
    [--locators=value] [--locator-wait-time=value] [--log-level=value]
    [--mcast-address=value] [--mcast-port=value] [--memcached-port=value]
    [--memcached-protocol=value] [--rebalance(=value)] [--server-bind-address=value]
    [--server-port=value] [--spring-xml-location=value]
    [--statistic-archive-file=value] [--dir=value] [--initial-heap=value]
    [--max-heap=value] [--use-cluster-configuration(=value)] [--J=value(,value)*]
    [--critical-heap-percentage=value] [--critical-off-heap-percentage=value]
    [--eviction-heap-percentage=value] [--eviction-off-heap-percentage=value]
    [--hostname-for-clients=value] [--max-connections=value]
    [--message-time-to-live=value] [--max-message-count=value] [--max-threads=value]
    [--socket-buffer-size=value] [--lock-memory=value] [--off-heap-memory-size=value]
    [--start-rest-api=value]
    [--http-service-port=value] [--http-service-bind-address=value]
    [--user=value] [--password=value]
```

Parameters, start server

Name	Description	Default Value
--name	Member name for this server. If not specified, gfsh generates a random name.	
--assign-buckets	Whether to assign buckets to the partitioned regions of the cache on server start.	false

Name	Description	Default Value
--bind-address	The IP address on which the server will be bound.	binds to all local addresses
--cache-xml-file	Specifies the name of the XML file or resource to initialize the cache with when it is created.	
--classpath	Application classes to be added to the server's CLASSPATH after the core jar file. See Setting Up the CLASSPATH for details.	
--include-system-classpath	When true, include the System CLASSPATH on the Server's CLASSPATH, as the System CLASSPATH is not included by default. If specified without a value, the value is set to true.	false
--disable-default-server	Whether the cache server will be started by default. If the parameter is specified without a value, the value is set to true. If set to true, the cache server acts as a peer.	false
--disable-exit-when-out-of-memory	Prevents the JVM from exiting when an OutOfMemoryError occurs.	false
--enable-time-statistics	Causes additional time-based statistics to be gathered for VMware GemFire operations.	true
--properties-file	The `gemfire.properties` file for configuring the server's cluster. The file's path can be absolute or relative to the gfsh working directory.	
--security-properties-file	The `gfsecurity.properties` file for configuring the server's security configuration in the cluster. The file's path can be absolute or relative to gfsh directory.	
--groups	Group(s) the Cache Server will be a part of.	
--force	Whether to allow the PID file from a previous Cache Server run to be overwritten.	false
--locators	Sets the list of locators used by the Cache Server to join the appropriate VMware GemFire cluster.	
--locator-wait-time	Sets the number of seconds the server will wait for a locator to become available during startup before giving up.	0
--log-level	Sets the level of output logged to the Cache Server log file. Possible values for log-level include: ALL, TRACE, DEBUG, INFO, WARN, ERROR, FATAL, OFF.	
--mcast-address	The IP address or hostname used to bind the UDP socket for multi-cast networking so the Cache Server can locate other members in the VMware GemFire cluster. If mcast-port is zero, then mcast-address is ignored.	
--mcast-port	Sets the port used for multi-cast networking so the Cache Server can locate other members of the VMware GemFire cluster. A zero value deactivates mcast.	

Name	Description	Default Value
--memcached-port	If specified and is non-zero, sets the port number for an embedded Gemcached server and starts the Gemcached server.	
--memcached-protocol	Sets the protocol used by an embedded Gemcached server. Valid values are `BINARY` and `ASCII`. If you omit this property, the ASCII protocol is used.	
--server-bind-address	Overrides the `bind-address` on which this server will listen for client connections. Set this option in a multi-homed server environment to distinguish communications from clients. Setting a value of the empty string ("") uses the value of `bind-address`.	value of `bind-address`
--server-port	Port the Server will listen on for client connections.	40404
--spring-xml-location	Specifies the location of a Spring XML configuration file(s) for bootstrapping and configuring a VMware GemFire Server. This configuration file can exist on the CLASSPATH (default) or any location supported by Spring's Resource(Loader) location specifiers (for example, classpath:, file:, etc). ResourceLoader is described in the Spring documentation.	
--rebalance	Whether to initiate rebalancing across the VMware GemFire cluster.	false
--dir	Specify the directory in which the server will run in. This directory is written to the location where you started `gfsh`.	If not specified, the directory is named after the server.
--statistic-archive-file	The file that statistic samples are written to. For example: "StatisticsArchiveFile.gfs". Must be defined to store the archiving to a file. An empty string (default) deactivates statistic archival.	not set
--initial-heap	Initial size of the heap in the same format as the JVM `-Xms` parameter.	
--max-heap	Maximum size of the heap in the same format as the JVM `-Xmx` parameter. **Note:** The additional GC parameters introduced by the `--max-heap` option are not compatible with the G1 garbage collector.	
--J	Argument passed to the JVM on which the Cache Server will run. For example, `--J=-Dfoo.bar=true` will set the property "foo.bar" to "true". If the argument you are passing contains spaces or commas, enclose the option in single quotes.	
--use-cluster-configuration	Specifies whether the server requests a cluster configuration from the locator. See Overview of the Cluster Configuration Service.	true

Name	Description	Default Value
--critical-heap-percentage	Set the percentage of heap at or above which the cache is considered in danger of becoming inoperable due to garbage collection pauses or out of memory exceptions. Past the threshold, operations that require heap space will throw a `LowMemoryException`. This feature requires additional VM flags to perform properly; you must set `--initial-heap` and `--max-heap` or the corresponding JVM properties to use this threshold. You must also set `--max-heap` and `--initial-heap` to the same value.	0 (no critical heap threshold enforced)
--critical-off-heap-percentage	The percentage of off-heap memory used at or above which the cache is considered in danger of becoming inoperable due to out of memory exceptions. Past the threshold, operations that require heap space will throw a `LowMemoryException`.	0 (no critical off-heap threshold enforced)
--eviction-heap-percentage	Set the percentage of heap at or above which the eviction should begin on Regions configured for HeapLRU eviction. Changing this value may cause eviction to begin immediately. Only one change to this attribute or critical heap percentage will be allowed at any given time and its effect will be fully realized before the next change is allowed. This feature requires additional VM flags to perform properly; you must set `--initial-heap` and `--max-heap` or the corresponding JVM properties to use this threshold. You must also set `--max-heap` and `--initial-heap` to the same value.	• 0, if no region is configured with heap eviction • If `critical-heap-percentage` is set to a non-zero value, 5% less than that value. • 80%, if `critical-heap-percentage` is not configured.
--eviction-off-heap-percentage	The percentage of off-heap memory used at or above which the eviction should begin on regions configured for off-heap and HeapLRU eviction. Changing this value may cause eviction to begin immediately. Only one change to this attribute or critical off-heap percentage will be allowed at any given time, and its effect will be fully realized before the next change is allowed.	• 0, if no region is configured with heap eviction • If `critical-off-heap-percentage` is set to a non-zero value, 5% less than that value. • 80%, if `critical-off-heap-percentage` is not configured.

Name	Description	Default Value
--hostname-for-clients	Sets the IP address or host name that a locator will provide to clients. Clients use the address to connect to a server. Set this value when clients use a different address to connect with the server than the `bind-address`, as those clients might with servers in a private cloud or multi-homed environment. Not specifying this option or setting this option to the empty string ("") causes the `bind-address` to be given to clients.	
--max-connections	Sets the maximum number of client connections allowed. When the maximum is reached the cache server will stop accepting connections.	
--message-time-to-live	Sets the time (in seconds) after which a message in the client queue will expire.	
--max-message-count	Sets maximum number of messages that can be enqueued in a client-queue.	
--max-threads	Sets the maximum number of threads allowed in this cache server to service client requests. The default of 0 causes the cache server to dedicate a thread for every client connection. When client-server TLS/SSL is configured, values other than the default are not supported.	
--socket-buffer-size	Sets the buffer size in bytes of the socket connection for this CacheServer. The default is 32768 bytes.	
--lock-memory	(Linux only) When true, the member's heap and off-heap memory are locked in RAM, preventing them from being paged to disk. You must increase the related `ulimit` operating system resource to allow the OS to lock memory chunks of sufficient size.	false
--off-heap-memory-size	The integer quantity of off-heap memory to be used for storing region values. Specified in Gigabytes with a 'g' suffix, or Megabytes with an 'm' suffix. For example, allocate a 2 Gigabyte off-heap space with `--off-heap-memory-size=2g`. The default value of 0 does not use any off-heap memory.	0
--start-rest-api	When true, starts the REST API service.	false
--redirect-output	When true, redirect standard output and standard error to the server log file. If specified without a value, the value is set to true.	false
--http-service-port	Specifies the HTTP service port.	7070
--http-service-bind-address	Specifies the IP address to which the HTTP service will be bound.	the local host machine's address
--user	The user name of the credential to use in authenticating to the cluster. When specified, if the `--password` option is not also specified, then `gfsh` will prompt for the password.	

Name	Description	Default Value
--password	The password portion of the credential to use in authenticating to the cluster.	

Examples

```
gfsh>start server --name=server1
gfsh>start server --name=server2 --server-port=40405
```

start vsd

Launch GemFire Visual Statistics Display (VSD) in a separate process.

You can specify a comma delimited list of directories and specific GemFire statistics archive files (.gfs) to load into VSD upon start. A directory locations will be searched recursively for statistics archive (.gfs) files.

Availability: Online or offline.

Syntax:

```
start vsd [--file=value(nullvalue)*]
```

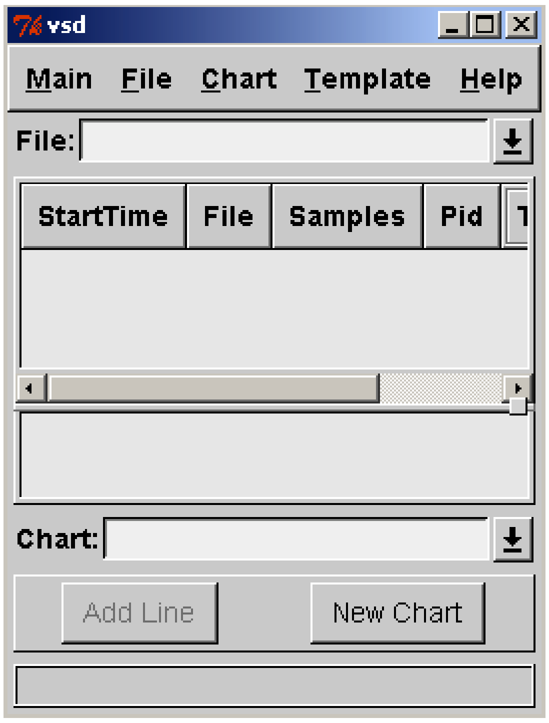
Parameters, start vsd

Name	Description
–file	File or directory from which to read the statistics archive(s).

Example Commands:

```
gfsh> start vsd /export/gemfire/node1,/export/gemfire/node2/statArchive.gfs,../../gemf
ire/nodeN;
Running GemFire Visual Statistics Display (VSD)

gfsh> start vsd;
Running GemFire Visual Statistics Display (VSD)
```

Sample Output:



status

Check the status of the cluster configuration service and VMware GemFire member processes, including locators, gateway receivers, gateway senders, and servers.

- **status cluster-config-service**

  Displays the status of the cluster configuration service.

- **status gateway-receiver**

  Display the status of the specified gateway receiver.

- **status gateway-sender**

  Display the status of the specified gateway sender.

- **status locator**

  Displays the status of the specified locator.

- **status server**

  Display the status of the specified VMware GemFire cache server.

status cluster-config-service

Displays the status of the cluster configuration service.

Displays the status of cluster configuration service on all the locators where enable-cluster-configuration is set to `true`.

Availability: Online. You must be connected in `gfsh` to a JMX Manager member to use this command.

Syntax:

```
status cluster-config-service
```

Example Commands:

```
status cluster-config-service
```

Sample Output:

```
gfsh>status cluster-config-service
Status of shared configuration on locators

  Name    | Status
--------- | -------
locator8  | RUNNING
```

status gateway-receiver

Display the status of the specified gateway receiver.

Availability: Online. You must be connected in `gfsh` to a JMX Manager member to use this command.

Syntax:

```
status gateway-receiver [--groups=value(,value)*] [--members=value(,value)*]
```

Name	Description
--groups	Group(s) of Gateway Receivers for which to display status.
--members	Name or ID of the Gateway Receiver(s) for which to display status.

Table 1. Status Gateway-Receiver Parameters

Example Commands:

```
status gateway-receiver --groups=LN-Group1
status gateway-receiver --members=server1
```

Sample Output:

```
gfsh>status gateway-receiver
Member                | Port   | Status
---------------------| ------| -------
pc13(8151)<v2>:26518 | 26837 | Running
pc13(8175)<v4>:53787 | 23753 | Running
pc13(8164)<v3>:24081 | 25457 | Running

Member                        | Error
------------------------------| ---------------------------------------------------
pc13(Manager:8124)<v1>:52410 | GatewayReceiver is not available or already stopped
pc13(8130):8180              | GatewayReceiver is not available or already stopped


gfsh>status gateway-receiver --members=pc13(8151)<v2>:50130
```

```
Member               | Port  |  Status
-------------------- | ----- | --------
pc13(8151)<v2>:50130 | 28592 | Running

gfsh>status gateway-receiver --group=RG1
Member                 | Port   | Status
-----------------------| ------| -------
pc13(8151)<v2>:19450   | 27815 | Running
pc13(8175)<v4>:14139   | 27066 | Running
pc13(8164)<v3>:45150   | 29897 | Running
```

status gateway-sender

Display the status of the specified gateway sender.

Availability: Online. You must be connected in `gfsh` to a JMX Manager member to use this command.

Syntax:

```
status gateway-sender --id=value [--groups=value(,value)*]
[--members=value(,value)*]
```

Name	Description
--id	*Required*. ID of the Gateway Sender.
--groups	Group(s) of Gateway Senders for which to display status. Comma separated list for multiple member groups.
--members	Name/ID of the Gateway Sender(s) for which to display status.

Table 2. Status Gateway-Sender Parameters

Example Commands:

```
status gateway-receiver receiver1-LN --groups=LN-Group1;
```

Sample Output:

```
gfsh>status gateway-sender --id=ln_Serial
Member                 |  Type   | Runtime Policy | Status
-----------------------| -------| -------------- | -----------
 pc13(8175)<v4>:21449  | Serial | Secondary      | Not Running
 pc13(8151)<v2>:40761  | Serial | Secondary      | Not Running
 pc13(8164)<v3>:33476  | Serial | Secondary      | Not Running

Member                          | Error
------------------------------ | ------------------------------
 pc13(8130):2365                | GatewaySender is not available
 pc13(Manager:8124)<v1>:43821   | GatewaySender is not available

gfsh>status gateway-sender --id=ln_Serial --members=pc13(8151)<v2>:7411
Member               |  Type   | Runtime Policy | Status
-------------------  | ------ | -------------- | -----------
 pc13(8151)<v2>:7411 | Serial | Secondary      | Not Running
```

```
gfsh>status gateway-sender --id=ln_Serial --members=pc13(8151)<v2>:7411
Member               |  Type  | Runtime Policy | Status
-------------------- -| ------ | -------------- | -------
 pc13(8151)<v2>:7411 | Serial | Primary        | Running

gfsh>status gateway-sender --id=ln_Serial --groups=Serial_Sender
==>
Member                 |  Type  | Runtime Policy | Status
---------------------- | -------| -------------- | -----------
 pc13(8151)<v2>:44396  | Serial | Secondary      | Not Running
 pc13(8164)<v3>:29475  | Serial | Secondary      | Not Running
Member                 | Error
---------------------- | ------------------------------
 pc13(8186)<v5>:45840  | GatewaySender is not available
```

status locator

Displays the status of the specified locator.

The status will be one of the following:

- started
- online
- offline
- not responding

Availability: Online or offline. If you want to obtain the status of a locator while you are offline, use the `--dir` option.

Syntax:

```
status locator [--name=value] [--host=value] [--port=value] [--dir=value] [--security-
properties-file=value]
```

Name	Description	Default Value
--name	Name/ID of the locator for which to display status. You must be connected to the JMX Manager to use this option. Can be used to obtain status of remote locators. See Using gfsh to Manage a Remote Cluster Over HTTP or HTTPS.	
--host	Hostname or IP address on which the Locator is running.	
--port	Port on which the locator is listening.	10334
--dir	Directory in which the locator was started.	current directory
--security-properties-file	The properties file for configuring SSL to connect to the SSL-enabled Locator. The file's path can be absolute or relative to the gfsh directory.	current directory

Table 3. Status Locator Parameters

Example Commands:

```
status locator
status locator --name=locator1
```

status server

Display the status of the specified VMware GemFire cache server.

Availability: Online or offline. If you want to obtain the status of a server while you are offline, use the `--dir` option.

Syntax:

```
status server [--name=value] [--dir=value]
```

Name	Description	Default Value
--name	Name or ID of the Cache Server for which to display status. You must be connected to the JMX Manager to use this option. Can be used to obtain status of remote servers. See Using gfsh to Manage a Remote Cluster Over HTTP or HTTPS.	
--dir	Directory in which the VMware GemFire Cache Server was started.	current directory

Table 4. Status Server Parameters

Example Commands:

```
status server
status server --name=server1
```

stop

Stop gateway receivers, gateway senders, locators and servers.

- **stop gateway-receiver**

  Stop the gateway receiver on a member or members.

- **stop gateway-sender**

  Stop a gateway sender with a given id on a specified member or members of a specified member group.

- **stop locator**

  Stop a locator.

- **stop server**

  Stop a VMware GemFire cache server.

stop gateway-receiver

Stop the gateway receiver on a member or members.

Availability: Online. You must be connected in `gfsh` to a JMX Manager member to use this command.

Syntax:

```
stop gateway-receiver [--groups=value(,value)*] [--members=value(,value)*]
```

Name	Description
--groups	Group(s) of members on which to stop the Gateway Receiver. Use a comma-separated list for multiple member groups.
--members	Name/Id of the member(s) on which to stop the Gateway Receiver.

Table 1. Stop Gateway-Receiver Parameters

Example Commands:

```
stop gateway-receiver --members=receiver1-LN
stop gateway-receiver --groups=LN-Group1
```

Sample Output:

```
gfsh>stop gateway-receiver
      Member                | Result | Message
--------------------------- | -------| -----------------------------------------------
------------------------
pc13(2266)<v6>:56852        | OK     | GatewayReceiver is stopped on member pc13(2266)
<v6>:56852
pc13(Manager:2242)<v5>:57631| Error  | GatewayReceiver is not available on member pc13
(Manager:2242)<v5>:57631
pc13(2275)<v7>:47480        | OK     | GatewayReceiver is stopped on member pc13(2275)
<v7>:47480
pc13(2293)<v8>:55472        | OK     | GatewayReceiver is stopped on member pc13(2293)
<v8>:55472

gfsh>stop gateway-receiver --members=pc13(2266)<v14>:36579
GatewayReceiver is stopped on member pc13(2266)<v14>:36579

gfsh>stop gateway-receiver --groups=RG1
         Member      | Result | Message
-------------------- | -------| -----------------------------------------------------
----
pc13(2275)<v23>:27484| OK     | GatewayReceiver is stopped on member pc13(2275)<v23>:2
7484
pc13(2293)<v24>:55810| OK     | GatewayReceiver is stopped on member pc13(2293)<v24>:5
5810
pc13(2266)<v22>:4522 | OK     | GatewayReceiver is stopped on member pc13(2266)<v22>:4
522
```

stop gateway-sender

Stop a gateway sender with a given id on a specified member or members of a specified member group.

CAUTION: Use caution with the `stop gateway-sender` command (or equivalent `GatewaySender.stop()` API) on parallel gateway senders. Instead of stopping an individual parallel gateway sender on a member, we recommend shutting down the entire member to ensure that proper failover of partition region events to other gateway sender members. Using this command on an individual parallel gateway sender can occur in event loss. See Stopping Gateway Senders for more details.

Availability: Online. You must be connected in `gfsh` to a JMX Manager member to use this command.

Syntax:

```
stop gateway-sender --id=value [--groups=value(,value)*] [--members=value(,value)*]
```

Name	Description
--id	*Required.* ID of the Gateway Sender.
--groups	Group(s) of members on which to stop the Gateway Sender.
--members	Name/Id of the member(s) on which to stop the Gateway Sender.

Table 2. Stop Gateway-Sender Parameters

Example Commands:

```
stop gateway-sender --id=ln --members=server1
```

Sample Output:

```
gfsh>stop gateway-sender --id=ln
Member                        | Result | Message
----------------------------- | ------ | ------------------------------------------------
-------------------------
pc13(5184)<v7>:41914          | OK     | GatewaySender ln is stopped on member pc13(518
4)<v7>:41914
pc13(5192)<v8>:25704          | OK     | GatewaySender ln is stopped on member pc13(519
2)<v8>:25704
pc13(5174)<v6>:53996          | OK     | GatewaySender ln is stopped on member pc13(517
4)<v6>:53996
pc13(Manager:5148)<v5>:64040 | Error  | GatewaySender ln is not available on member pc
13(Manager:5148)<v5>:64040

gfsh>stop gateway-sender --id=ln --members=pc13(5174)<v14>:17819
GatewaySender ln is stopped on member pc13(5174)<v14>:17819

gfsh>stop gateway-sender --id=ln --groups=SenderGroup1

Member                | Result | Message
--------------------- | ------ | -----------------------------------------------------
------
pc13(5174)<v18>:63332 | OK     | GatewaySender ln is stopped on member pc13(5174)<v18
>:63332
pc13(5184)<v19>:20055 | OK     | GatewaySender ln is stopped on member pc13(5184)<v19
>:20055
pc13(5192)<v20>:14622 | OK     | GatewaySender ln is stopped on member pc13(5192)<v20
>:14622
```

stop locator

Stop a locator.

Note: One of the command line options `--name` or `--dir` must be specified to identify the locator to be stopped.

Availability: Online or offline. If you want to stop a locator while you are offline, use the `--dir` option.

Syntax:

```
stop locator --name=value | --dir=value
```

Name	Description	Default Value
--name	The VMware GemFire member name of the locator to stop. You must be connected to the JMX Manager to use this option. Can be used to stop remote locators. See Using gfsh to Manage a Remote Cluster Over HTTP or HTTPS.	
--dir	Directory in which the locator was started.	current directory

Table 3. Stop Locator Parameters

Example Commands:

```
stop locator --name=locator3
```

Sample Output:

```
gfsh>stop locator --name=locator3
Stopping Locator running in /Users/test/locator3 on 192.0.2.0[10334] as locator3...
Process ID: 71531
Log File: /Users/test/locator3/locator3.log
...
No longer connected to 192.0.2.0[1099].
```

```
gfsh>stop locator --dir=loc2
Stopping Locator running in /Users/test/loc2 on 192.0.2.0[10334] as loc2...
Process ID: 71714
Log File: /Users/test/loc2/loc2.log
...
No longer connected to 192.0.2.0[1099].
```

stop server

Stop a VMware GemFire cache server.

Availability: Online or offline. If you want to stop a cache server while you are offline, use the `--dir` option.

Syntax:

```
stop server [--name=value] [--dir=value]
```

Name	Description	Default Value
--name	Name/Id of the VMware GemFire Cache Server to stop. You must be connected to the JMX Manager to use this option. Can be used to stop remote servers. See Using gfsh to Manage a Remote Cluster Over HTTP or HTTPS.	
--dir	Directory in which the VMware GemFire Cache Server was started.	current directory

Table 4. Stop Server Parameters

Example Commands:

```
stop server --name=server1
stop server --dir=server1
```

undeploy

Undeploy the JAR files that were deployed on members or groups using `deploy` command.

If `--jars` is not specified, the command will undeploy all deployed JARs. If `--groups` is not specified, the command applies to the entire cluster. Note that this command can't unload the classes that were loaded during deployment. Member(s) should be restarted for that.

Availability: Online. You must be connected in `gfsh` to a JMX Manager member to use this command.

Syntax:

```
undeploy [--jars=value(,value)*] [--groups=value(,value)*]
```

Name	Description	Default Value
--groups	Group(s) from which the specified JAR(s) will be undeployed.	undeploy will occur on all members
--jars	JAR or JARs to be undeployed.	All JARs will be undeployed

Table 1. Undeploy Parameters

Example Commands:

```
undeploy --jars=domain-objects.jar
undeploy --groups=Group1
```

Sample Output:

```
gfsh>undeploy --jars=domain-objects.jar

  Member    |  Un-Deployed JAR   |          Un-Deployed From JAR Location
---------- | ------------------ | ---------------------------------------------
datanode10 | domain-objects.jar | /usr/local/gemfire/deploy/GF#domain-objects#1
datanode11 | domain-objects.jar | /usr/local/gemfire/deploy/GF#domain-objects#1
```

```
gfsh> undeploy --groups=Group1

 Member   |     Un-Deployed JAR     |             Un-Deployed From JAR Location
--------- | ----------------------- | -------------------------------------------------
------
datanode1 | group1_functions.jar    | /usr/local/gemfire/deploy/GF#group1_functions.ja
r#1
datanode1 | group1_dependencies.jar | /usr/local/gemfire/deploy/GF#group1_dependencie
s.jar#1
datanode2 | group1_functions.jar    | /usr/local/gemfire/deploy/GF#group1_functions.ja
r#1
datanode2 | group1_dependencies.jar | /usr/local/gemfire/deploy/GF#group1_dependencie
s.jar#1
```

Error Messages:

```
No JAR Files Found
```

validate offline-disk-store

Validate offline disk stores.

Availability: Offline.

Syntax:

```
validate offline-disk-store --name=value --disk-dirs=value(,value)*
```

Name	Description
--name	*Required*. Name of the disk store to be validated.
--disk-dirs	*Required.* Directories where data for the disk store was previously written.

Table 1. Validate Offline-Disk-Store Parameters

Example Commands:

```
validate offline-disk-store --name=DiskStore2 --disk-dirs=data/dir3,data/dir4
```

version

Display product version information.

Availability: Online or offline.

Syntax:

```
version [--full]
```

Name	Description	Defalut Value
--full	Show the full version information.	false

Table 1. Version Parameters

Example Commands:

```
version
version --full
```

Sample Output:

```
gfsh>version
v8.0.0


gfsh>version --full
Java version:   8.0.0 build 48319 07/31/2014 17:26:09 PDT javac 1.8.0_1
Native version: native code unavailable
Source revision: 48319
Source repository: gemfire/branches/cedar_dev_Oct12
Running on: /192.0.2.0, 1 cpu(s), amd64 Linux 2.6.32-38-generic
```

Creating and Running gfsh Command Scripts

gfsh offers several ways to run commands in a scripting environment.

Running gfsh Scripts

You can create and run scripts that contain gfsh commands that you wish to execute. To execute the script, use the gfsh run command. For example:

```
gfsh run --file=mycommands.gfsh
```

Note: When you run a gfsh script, interactive parameters are ignored. You can also set the script to run in quiet mode to prevent output and instruct the script to skip any errors it encounters.

Your command history file can be helpful when you write a gfsh script. A history of commands that have been executed successfully is logged in the `.gfsh.history` file in the home directory of the user running gfsh. You can also export a history file with the `history --file=your_file_name` command.

When a user runs `start server` or `start locator` from gfsh without specifying the member name, gfsh will automatically pick a random member name. This is useful for automation.

Running gfsh Commands on the OS Command Line

You can run some `gfsh` commands directly from your operating system's prompt by preceding the command with `gfsh`. This can be useful for Unix shell or Windows batch scripting. For example:

```
$ gfsh start locator --name=locator2 --port=10335
```

To see which gfsh commands are available directly on the prompt:

```
$ gfsh help
```

Running Multiple gfsh Commands on the OS Command Line

To run multiple commands directly on the command line, use the `-e` option followed by the `gfsh` command within quote marks. For example:

```
prompt>gfsh -e "start locator --name=locator1" -e "start server --name=server1"

prompt>gfsh -e "start jconsole"

prompt>gfsh -e "connect --locator=remotehost[10334]" -e "rebalance" -e "gc"
```

Mapping cache.xml Elements to gfsh Configuration Commands

You can configure a VMware GemFire cluster using either cache.xml files, or you can use gfsh and the cluster configuration service to configure a cluster. This table maps `cache.xml` elements to the gfsh commands that configure and manage a cluster.

cache.xml Element	gfsh Command
<cache>, <cache-server>	• start server • status server • stop server • alter runtime
<async-event-queue>	• alter async-event-queue • create async-event-queue • destroy async-event-queue • list async-event-queues • resume async-event-queue-dispatching
<pdx>	• configure pdx
<region>	• create region • alter region • destroy region • describe region • list regions • rebalance
<index>	• create index • destroy index • list indexes

cache.xml Element	gfsh Command
<disk-store>	• create disk-store • alter disk-store • backup disk-store • compact disk-store • compact offline-disk-store • describe disk-store • describe offline-disk-store • destroy disk-store • list disk-stores • revoke missing-disk-store • show missing-disk-stores • validate offline-disk-store
<query-config-service>	• alter query-service

Table 1. Migrating cache.xml elements to gfsh commands

Gemcached

Gemcached is a VMware GemFire adapter that allows Memcached clients to communicate with a VMware GemFire server cluster, as if the servers were memcached servers. Memcached is an open-source caching solution that uses a distributed, in-memory hash map to store key-value pairs of string or object data.

For information about Memcached, see http://www.memcached.org.

- **How Gemcached Works**

  Applications use memcached clients to access data stored in embedded Gemcached servers.

- **Deploying and Configuring a Gemcached Server**

  You can configure and deploy Gemcached servers in a Java class or by using the gfsh command-line interface.

- **Advantages of Gemcached over Memcached**

  The standard memcached architecture has inherent architectural challenges that make memcached applications difficult to write, maintain, and scale. Using Gemcached with VMware GemFire addresses these challenges.

How Gemcached Works

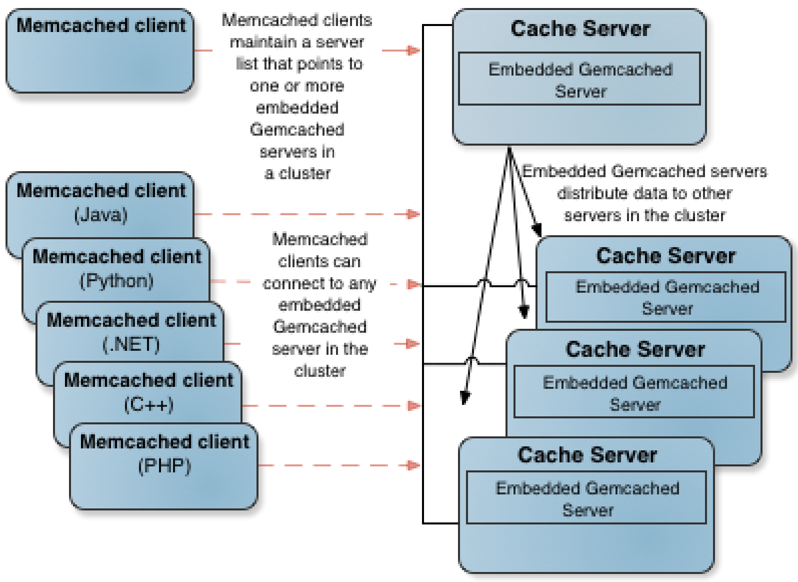
Applications use memcached clients to access data stored in embedded Gemcached servers.

Applications can use memcached clients that are written in Python, C#, Ruby, PHP, and other programming languages. Each memcached server in a cluster stores data as key/value pairs. A memcached client maintains a list of these servers, determines which server has the required data, and accesses the data directly on that server.

To integrate memcached with VMware GemFire, you embed a Gemcached server within a VMware GemFire cache server. These *Gemcached* servers take the place of memcached servers. The memcached client uses its normal wire protocol to communicate with the Gemcached servers, which appear to the client as memcached servers. No code changes in the clients are needed. VMware GemFire manages the distribution and access to data among the embedded Gemcached servers.

As shown in Gemcached Architecture, memcached clients, which ordinarily maintain a list of memcached servers, now maintain a list of embedded Gemcached servers. If more embedded Gemcached servers are added to the cluster, the new servers automatically become part of the cluster. The memcached clients can continue to communicate with the servers on the list, without having to update their list of servers.

Figure: Gemcached Architecture


Memcached client
Memcached clients maintain a server list that points to one or more embedded Gemcached servers in a cluster
Cache Server
Embedded Gemcached Server
Embedded Gemcached servers distribute data to other servers in the cluster
Memcached client (Java)
Memcached client (Python)
Memcached client (.NET)
Memcached client (C++)
Memcached client (PHP)
Memcached clients can connect to any embedded Gemcached server in the cluster
Cache Server
Embedded Gemcached Server
Cache Server
Embedded Gemcached Server
Cache Server
Embedded Gemcached Server

Memcached clients use the memcached API to read and write data that is stored in memcached servers; therefore, client-side VMware GemFire features are not available to these clients. Gemcached servers, however, can use VMware GemFire's server-side features and API. These features include the following. (For more detail, see Advantages of Gemcached over Memcached.)

- Data consistency and scalability.
- High availability.
- Read-through, write through, and write behind to a database, implemented from within the distributed VMware GemFire cache.
- Storage keys and values of any type and size.
- For applications, a choice among partitioned and replicated region configurations.
- Automatic overflow of data to disk in low-memory scenarios.
- Efficient persistence of data to disk.
- Configurable expiration of cached data.

- Configurable eviction of cached data.

Deploying and Configuring a Gemcached Server

You can configure and deploy Gemcached servers in a Java class or by using the gfsh command-line interface.

The following sections describe how to configure and deploy an embedded Gemcached server. You can configure and start a GemCached server either by invoking a Java class that calls the cache server's `start()` method, or by starting the cache server using the gfsh command line.

Embedding a Gemcached server in a VMware GemFire Java Application

The `org.apache.geode.memcached` package contains a single class, `GemFireMemcachedServer` (see the VMware GemFire Javadocs.) Use this class to configure and embed a Gemcached server in a VMware GemFire cache server. For example, the following statement creates and starts an embedded Gemcached server on port number 5555 using the binary protocol:

```
GemFireMemcachedServer server =
  new GemFireMemcachedServer(5555, Protocol.BINARY);server.start();
```

Note: By default, Gemcached servers use the ASCII protocol.

When you start a Gemcached server, by default, it creates a `RegionShortcut.PARTITION` region named `gemcached` where data used by memcached clients is stored. You can alter these defaults by configuring the region using the `cache.xml` or `gemfire.properties` files. See Distributed System and Cache Configuration.

Starting a Gemcached Server Using a gfsh Command

You can also start a Gemcached server with the gfsh command-line interface. Use the following syntax:

```
gfsh>start server
  --name=<server_name>
  --server-port=<port_number>
  --memcached-port=<port_number>
  --memcached-protocol=BINARY|ASCII
```

Note: You can also set the memcached port number and protocol in the `gemfire.properties` file.

If the `memcached-port` property is not specified, the embedded Gemcached server is not started.

Configuring a Gemcached Server with the gemfire.properties File

You can set the following properties in the `gemfire.properties` file that are used when starting Gemcached servers:

Table 1. Gemcached Properties

Property	Description
`memcached-port`	The port number where the Gemcached server listens for connections from memcached clients. If the port number is set to 0 or the `memcached-port` parameter is omitted, the Gemcached server does not start.
`memcached-protocol`	Memcached supports both ASCII and binary communication protocols. (See Memcached protocol By default, Gemcached uses the ASCII protocol. Set one of the following values: • `ASCII` (default) • `BINARY`

Table 1. Gemcached Properties

Advantages of Gemcached over Memcached

The standard memcached architecture has inherent architectural challenges that make memcached applications difficult to write, maintain, and scale. Using Gemcached with VMware GemFire addresses these challenges.

Data consistency. Memcached clients must maintain a list of servers where the distributed data is stored. Each client must maintain an identical list, with each list ordered in the same way. It is the responsibility of the application logic to maintain and propagate this list. If some clients do not have the correct list, the client can retrieve stale data. In VMware GemFire clusters, all members communicate with each other to maintain data consistency, which eliminates the need to code these behaviors in the memcached clients.

High availability. When a memcached server becomes unavailable, memcached clusters are subject to failures or degraded performance because clients must directly query the backend database. Memcached-based applications must be coded to handle these failures, while VMware GemFire clusters handle such failures natively.

Faster cluster startup time. When a memcached cluster fails and a restart is required, the data must be reloaded and distributed to the cluster members while simultaneously processing requests for data. These startup activities can be time-consuming. When a VMware GemFire cluster restarts, data can be reloaded from other in-memory, redundant copies of the data or from disk, without having to query the back end database.

Better handling of network segmentation. Large deployments of memcached can use hundreds of servers to manage data. If, due to network segmentation, some clients cannot connect to all nodes of a partition, the clients will have to fetch the data from the backend database to avoid hosting stale data. VMware GemFire clusters handle network segmentation to ensure that client responses are consistent.

Automatic scalability. If you need to add capacity to a memcached cluster, you must propagate a new server list to all clients. As new clients come on line with the new list, older clients may not have a consistent view of the data in the cluster, which can result in inconsistent data in the servers. Because new VMware GemFire cache server members automatically discover each other, memcached clients do not need to maintain a complete server list. You can add capacity simply by adding servers.

Scalable client connections. A memcached client may need to access multiple pieces of data stored on multiple servers, which can result in clients having a TCP connection open to every server. When a memcached client accesses a Gemcached server, only a single connection to a Gemcached server instance is required. The Gemcached server manages the distribution of data using VMware GemFire's standard features.

HTTP Session Management Modules

The VMware GemFire HTTP Session Management modules provide fast, scalable, and reliable session replication for HTTP servers without requiring application changes.

VMware GemFire offers HTTP session management modules for tc Server, Tomcat, and AppServers.

These modules are included with the VMware GemFire product distribution, and installation .zip files can be found in the `tools/Modules` directory of your product installation.

- **HTTP Session Management Quick Start**

  In this section you download, install, and set up the HTTP Session Management modules.

- **Advantages of Using VMware GemFire for Session Management**

  The HTTP Session Management Module enables you to customize how you manage your session data.

- **Common Topologies for HTTP Session Management**

  Decide which topology is best for your usage. The module configuration process is slightly different for each topology.

- **General Information on HTTP Session Management**

  This section provides information on sticky load balancers, session expiration, additional VMware GemFire property changes, serialization and more.

- **Session State Log Files**

  Several log files are written by the various parts of the session management code.

- **Configuring Non-Sticky Sessions**

  This section describes the configuration of non-sticky sessions.

- **HTTP Session Management Module for Pivotal tc Server**

  This section describes how to set up and use the HTTP session management module with tc Server templates.

- **HTTP Session Management Module for Tomcat**

  You set up and use the module by modifying the Tomcat's `server.xml` and `context.xml` files.

- **HTTP Session Management Module for AppServers**

  You implement session caching with the HTTP Session Management Module for AppServers with a special filter, defined in the `web.xml`, which is configured to intercept and wrap all requests.

HTTP Session Management Quick Start

In this section you download, install, and set up the HTTP Session Management modules.

Following the Apache Tomcat convention, this page assumes the CATALINA_HOME environment variable is set to the root directory of the “binary” Tomcat distribution. For example, if Apache Tomcat is installed in `/usr/bin/apache-tomcat-9.0.50` then

```
CATALINA_HOME=/usr/bin/apache-tomcat-9.0.50
```

Quick Start Instructions

1. Download and install one of the application servers.

Supported Application Server	Version	Download Location
tc Server	3.2	https://network.pivotal.io/products/pivotal-tcserver
Tomcat	8.5	Tomcat 8 Software Downloads
Tomcat	9.0	Tomcat 9 Software Downloads

   The generic HTTP Session Management Module for AppServers is implemented as a servlet filter and should work on any application server platform that supports the Java Servlet 3.1 specification.

2. The HTTP Session Management Modules installation .zip files are located in the `tools/Modules` directory of the product installation directory. Locate the .zip file for the HTTP Session Management Module that you wish to install. Unzip the appropriate HTTP Session Management Module into the specified directory for your application server:

Supported Application Server	Version	Module	Target Location for Module
tc Server	2.9	Apache_Geode_Modules-SERVER-VERSION-tcServer.zip	`<tc Server root dir>/templates`
tc Server	3.2	Apache_Geode_Modules-SERVER-VERSION-tcServer30.zip	`<tc Server root dir>/templates`
Tomcat	8.5, 9.0	Apache_Geode_Modules-SERVER-VERSION-Tomcat.zip	`$CATALINA_HOME`

3. Complete the appropriate set up instructions for your application server described in the following sections:

   - Additional Quick Start Instructions for tc Server Module
   - Additional Quick Start Instructions for Tomcat Module

- Additional Instructions for AppServers Module

Additional Quick Start Instructions for tc Server Module

These steps provide a basic starting point for using the tc Server module. For more configuration options, see HTTP Session Management Module for Pivotal tc Server. As a prerequisite, module set up requires a JAVA_HOME environment variable set to the java installation.

1. Navigate to the root directory of tc Server.
2. Create a VMware GemFire instance using one of the provided templates and start the instance after starting up a locator. For example:

   ```
   $ gfsh start locator --name=locator1
   $ ./tcruntime-instance.sh create my_instance_name --template geode-p2p
   $ ./tcruntime-ctl.sh my_instance_name start
   ```

   This will create and run a VMware GemFire instance using the peer-to-peer topology and default configuration values. Another VMware GemFire instance on another system can be created and started in the same way.

   If you need to pin your tc Server instance to a specific tc Server runtime version, use the `--version` option when creating the instance.

Additional Quick Start Instructions for Tomcat Module

These steps provide a basic starting point for using the Tomcat module. For more configuration options, see HTTP Session Management Module for Tomcat.

1. Modify Tomcat's `server.xml` and `context.xml` files. Configuration is slightly different depending on the topology you are setting up and the version of Tomcat you are using.

   For example, in a peer-to-peer configuration using Tomcat 9, you would add the following entry within the `<server>` element of server.xml:

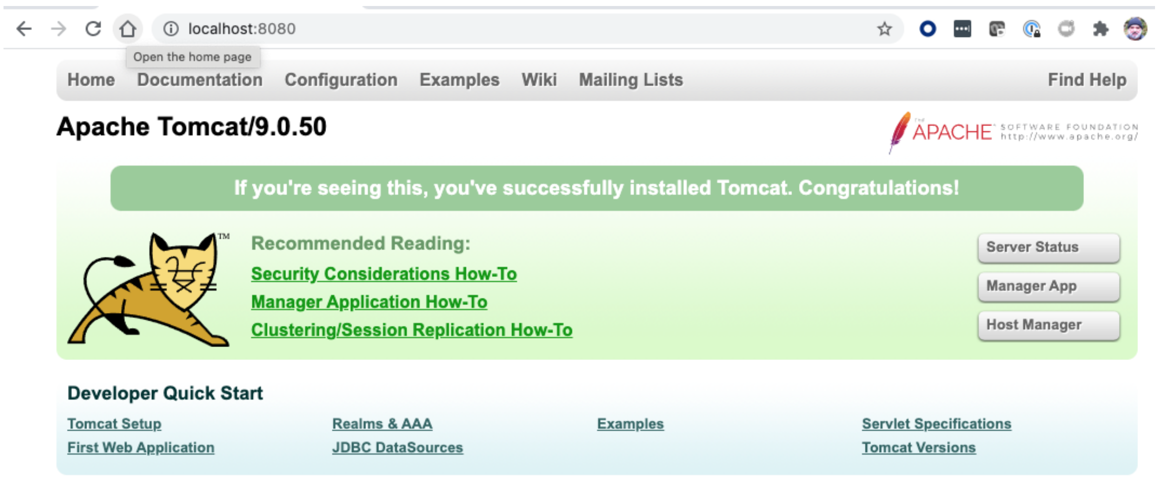
   ```
   <Listener className="org.apache.geode.modules.session.catalina.PeerToPeerCacheL
   ifecycleListener"
       locator="localhost[10334]" />
   ```

   and the following entry within the `<context>` tag in the context.xml file:

   ```
   <Manager className="org.apache.geode.modules.session.catalina.Tomcat9DeltaSessi
   onManager"/>
   ```

   See Setting Up the HTTP Module for Tomcat for additional instructions.
2. Start the Tomcat application server.

   ```
   $CATALINA_HOME/bin/startup.sh
   ```
3. Confirm that Tomcat is running by opening a browser and navigating to `localhost:8080`. If you see the Tomcat home page, your installation was successful.



Additional Instructions for AppServers Module

These steps provide a basic starting point for using the AppServers module with WebLogic, WebSphere or JBoss. For more configuration options, see HTTP Session Management Module for AppServers.

Note:

- The `modify_war` script relies upon a GEODE environment variable. Set the GEODE environment variable to the VMware GemFire product directory; this is the parent directory of `bin`.
- The `modify_war` script, described below, relies on files within the distribution tree and should not be run outside of a complete distribution.
- The `modify_war` script is a `bash` script and does not run on Windows.

To set up the AppServers module, perform the following steps:

1. Run the `modify_war` script against an existing `.war` or `.ear` file to integrate the necessary components. The example below will create a configuration suitable for a peer-to-peer VMware GemFire system, placing the necessary libraries into `WEB-INF/lib` for wars and `lib` for ears and modifying any `web.xml` files:

   ```
   $ bin/modify_war -w my-app.war -p gemfire.property.locators=localhost[10334] \
                    -t peer-to-peer
   ```

2. A new war file will be created called `session-my-app.war`. This file can now be deployed to the server.

Advantages of Using VMware GemFire for Session Management

The HTTP Session Management Module enables you to customize how you manage your session data.

Depending on your usage model, the HTTP Session Management Module enables you to accomplish the following tasks:

- Replicate session data across multiple peers.
- Partition data across multiple servers.
- Manage your session data in many other customizable ways.

Using VMware GemFire for session management has many advantages:

tc Server integration
: The VMware GemFire module offers clean integration into the tc Server environment with minimal configuration changes necessary.

Scalability
: Applications with a small number of frequently-accessed sessions can **replicate** session information on all members in the system. However, when the number of concurrent sessions being managed is large, data can be **partitioned** across any number of servers (either embedded within your application server process or managed by VMware GemFire cache servers), which allows for **linear scaling**. Additionally, capacity can be **dynamically added or removed** in a running system and VMware GemFire re-executes a non-blocking, rebalancing logic to migrate data from existing members to the newly added members. When the session state memory requirements exceed available memory, each partition host can **overflow to disk**.

Server-managed session state
: Session state can be managed independent of the application server cluster. This allows applications or servers to come and go without impacting session lifetimes.

Shared nothing cluster-wide persistence
: Session state can be persisted (and recovered) - invaluable for scenarios where sessions manage critical application state or have long lifetimes. VMware GemFire uses a shared nothing persistence model where each member can continuously append to rolling log files without ever needing to seek on disk, providing very high disk throughput. When data is partitioned, the total disk throughput can come close to the aggregate disk transfer rates across each of the members storing data on disk.

Session deltas
: When session attributes are updated, only the updated state that is sent over the wire (to cache servers and to replicas). This provides fast updates even for large sessions. Session state is always managed in a serialized state on the servers, avoiding the need for the cache servers to be aware of the application classes.

Tiered caching
: Applications can configure a local or near cache for in-process caching of the most frequently used session state. This local cache delegates to a farm of cache servers where the entire session state is partitioned across any number of peer cache servers. The local cache can be configured to consume a certain percentage of the total heap available before least-recently used (LRU) eviction. This is a simpler and more effective way to manage LRU caches as opposed to alternate strategies based on count or memory size, which increase the risk of getting an "OutOfMemoryException".

Application server sizing
: Another aspect of tiered-caching functionality is that session replication can be configured so that session objects are stored external to the application server process. This allows the heap settings on the application server to be much smaller than they would otherwise need to be.

High availability (HA), disk-based overflow, synchronization to backend data store, other VMware GemFire features

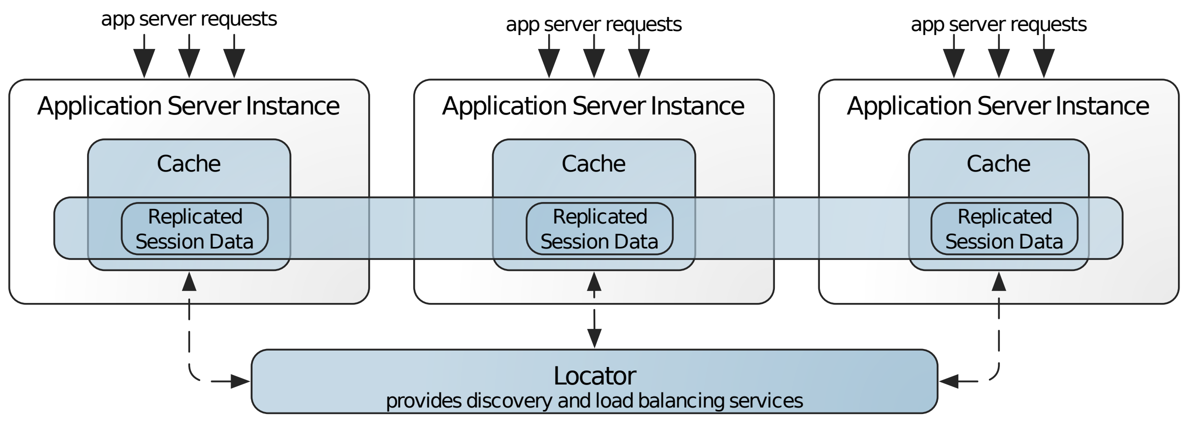
All the popular VMware GemFire features are available. For example: more than one synchronous copy of the session state can be maintained providing high availability (HA); the session cache can overflow to disk if the memory capacity in the cache server farm becomes insufficient; state information can be written to a backend database in a synchronous or asynchronous manner.

Common Topologies for HTTP Session Management

Decide which topology is best for your usage. The module configuration process is slightly different for each topology.

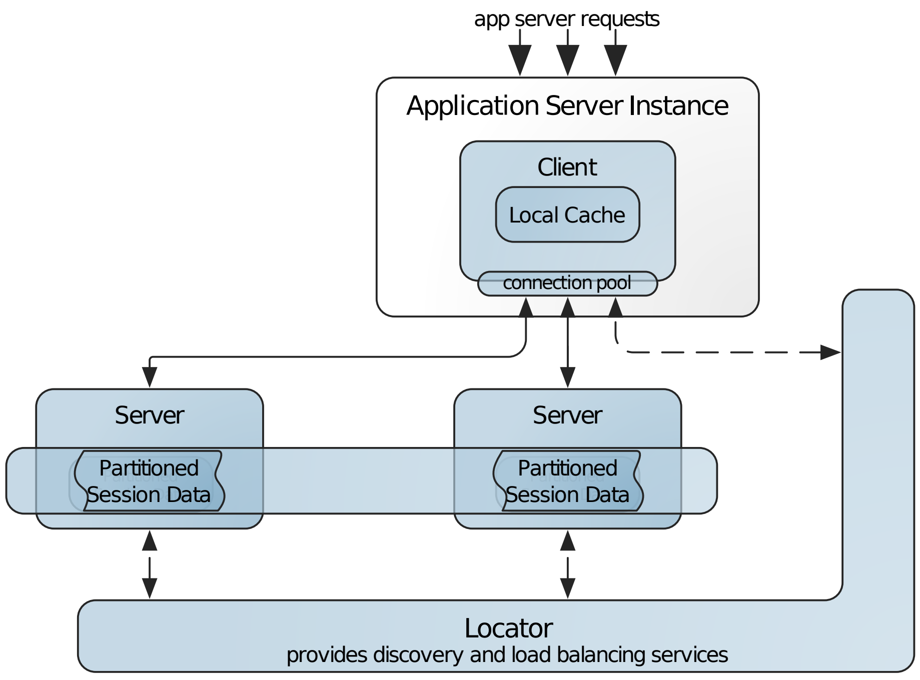
By default, a peer-to-peer configuration creates a replicated region across all servers. By default, a client/server configuration creates a partitioned region across all servers.

Peer-to-Peer Configuration



In a peer-to-peer configuration, each instance within an application server contains its own cache. Each instance communicates with the locator and with other instances as peers. By default, data is replicated. A peer-to-peer configuration is useful when multiple systems want fast access to all session data. This configuration is also the simplest one to set up and does not require any external processes.

Client/Server Configuration



In a client/server configuration, the Tomcat or tc Server instance operates as a VMware GemFire client, which must communicate with one or more VMware GemFire servers to acquire session data. The client maintains its own local cache and will communicate with the server to satisfy cache misses. A client/server configuration is useful when you want to separate the application server instance from the cached session data. In this configuration, you can reduce the memory consumption of the application server since session data is stored in separate VMware GemFire server processes.

General Information on HTTP Session Management

This section provides information on sticky load balancers, session expiration, additional VMware GemFire property changes, serialization and more.

Sticky Load Balancers

Typically, session replication will be used in conjunction with a load balancer enabled for sticky sessions. Sessions should be unique across application servers. With Tomcat, this can be accomplished by setting a JVM route (`JVMRoute=value`).

Session Expiration

To set the session expiration value, you must change the `session-timeout` value specified in your application server's `WEB-INF/web.xml` file. This value will override the VMware GemFire inactive interval, which is specified in Tomcat, for example, by `maxInactiveInterval` within `context.xml`.

When a session expires, it gets removed from the application server and from all VMware GemFire servers when running in client-server mode.

Making Additional VMware GemFire Property Changes

If you want to change additional VMware GemFire property values, refer to instructions on manually changing property values as specified in the VMware GemFire module documentation for Tomcat (Changing the Default VMware GemFire Configuration in the Tomcat Module) and Application Servers (Changing the Default VMware GemFire Configuration in the AppServers Module).

Module Version Information

To acquire VMware GemFire module version information, look in the web server's log file for a message similar to:

```
INFO: Initializing VMware GemFire Modules
Java version:   1.0.0 user1 041216 2016-11-12 11:18:37 -0700
          javac 1..0_272
Native version: native code unavailable
Source revision: 857bb75916640a066eb832b43b3c805f0dd7ed0b
Source repository: develop
Running on: /192.0.2.0, 8 cpu(s), x86_64 Mac OS X 10.11.4
```

Object Serialization

Objects managed by the HTTP Session Management Module must be serializable since the session's objects are serialized before being stored in the region.

Session State Log Files

Several log files are written by the various parts of the session management code.

- `catalina.log`. Log file written by the tc server
- `cacheserver.log`. Log file written by the VMware GemFire server process.
- `gemfire_modules.log`. Log file written by the VMware GemFire cache client.

Adding FINE Debug Logging to catalina.log

To add VMware GemFire-specific FINE logging to the `catalina.log` file, add the following lines to your `<instance>/conf/logging.properties` file:

```
org.apache.geode.modules.session.catalina.DeltaSessionManager.level = FINE
org.apache.geode.modules.session.catalina.DeltaSession.level = FINE
```

These configurations will add FINE logging to the `catalina.DATE.log` file. The following is an example of FINE logging:

```
06-Sep-2011 15:59:47.250 FINE org.apache.geode.modules.session.catalina.DeltaSessionMa
nager.start
DeltaSessionManager[container=StandardEngine[Catalina].StandardHost[localhost].Standar
dContext[/manager];
regionName=gemfire_modules_sessions; regionAttributesId=PARTITION_REDUNDANT]: Starting

06-Sep-2011 15:59:47.254 FINE org.apache.geode.modules.session.catalina.DeltaSessionMa
```

```
nager.registerJvmRouteBinderValve
DeltaSessionManager[container=StandardEngine[Catalina].StandardHost[localhost].Standar
dContext[/manager];
regionName=gemfire_modules_sessions; regionAttributesId=PARTITION_REDUNDANT]: Register
ing JVM route binder valve

06-Sep-2011 15:59:47.351 FINE org.apache.geode.modules.session.catalina.ClientServerSe
ssionCache.createOrRetrieveRegion
Created session region: org.apache.geode.internal.cache.LocalRegion[path='/gemfire_mod
ules_sessions';
scope=LOCAL';dataPolicy=EMPTY; gatewayEnabled=false]
```

Add Session State Logging to the VMware GemFire Server Log

To add session-state-specific logging to the VMware GemFire server log file, add the following property to the `catalina.properties` file for the tc Server instance:

```
geode-cs.enable.debug.listener=true
```

Adding this configuration will print logging in the server log such as the following:

```
[info 2011/09/06 15:18:27.749 PDT  <ServerConnection on port 40404 Thread 3> tid=0x32]
DebugCacheListener: Received
CREATE for key=5782ED83A3D9F101BBF8D851CE4E798E; value=DeltaSession[id=5782ED83A3D9F10
1BBF8D851CE4E798E;
sessionRegionName=gemfire_modules_sessions; operatingRegionName=unset]

[info 2011/09/06 15:18:27.769 PDT  <ServerConnection on port 40404 Thread 3> tid=0x32]
DebugCacheListener: Received UPDATE
for key=5782ED83A3D9F101BBF8D851CE4E798E; value=DeltaSession[id=5782ED83A3D9F101BBF8D8
51CE4E798E;
sessionRegionName=gemfire_modules_sessions; operatingRegionName=unset]

[info 2011/09/06 15:19:36.729 PDT  <Timer-2> tid=0x24] DebugCacheListener: Received EX
PIRE_DESTROY for
key=5782ED83A3D9F101BBF8D851CE4E798E
```

Adding Additional Debug Logging to the VMware GemFire Server Log

To add fine-level logging to the VMware GemFire cache server, add the ‘log-level’ property to the server process. For example:

```
gfsh> start server --name=server1 --cache-xml-file=../conf/cache-server.xml
              --log-level=fine
```

This will add fine-level logging to the `server.log` file.

Note: This will help debug VMware GemFire server issues, but it adds a lot of logging to the file.

Add Debug Logging to gemfire_modules.log

To add fine-level logging to the VMware GemFire Cache Client, add the 'log-level' property to the Listener element in the tc Server or Tomcat `server.xml` file. For example:

```
<Listener log-level="fine"
cache-xml-file="${geode-cs.cache.configuration.file}"
className="org.apache.geode.modules.session.catalina.ClientServerCacheLifecycleListene
r"
criticalHeapPercentage="${geode-cs.critical.heap.percentage}"
evictionHeapPercentage="${geode-cs.eviction.heap.percentage}"
log-file="${geode-cs.log.file}"
statistic-archive-file="${geode-cs.statistic.archive.file}"
statistic-sampling-enabled="${geode-cs.statistic.sampling.enabled}"/>
```

This will add fine-level logging to the file defined by the `${geode-cs.log.file}` property. The default log file name is `gemfire_modules.log`.

Note: This will help debug VMware GemFire client issues, but it adds a lot of logging to the file.

HTTP Session Management Module for Pivotal tc Server

This section describes how to set up and use the HTTP session management module with tc Server templates.

If you would prefer to manually change the `server.xml` and `context.xml` files rather than use tc Server templates, refer to HTTP Session Management Module for Tomcat.

- **Installing the HTTP Module for tc Server**

  This topic describes how to install the HTTP session management module with tc Server templates.

- **Setting Up the HTTP Module for tc Server**

  To set up the HTTP Module for tc Server, start a tc Server instance with the appropriate tc Server template based on your preferred topology.

- **Changing the Default VMware GemFire Configuration in the tc Server Module**

  By default, the tc Server HTTP module will run VMware GemFire automatically with pre-configured settings. You can change these VMware GemFire settings.

Installing the HTTP Module for tc Server

This topic describes how to install the HTTP session management module with tc Server templates.

1. If you do not already have tc Server, download and install the product from the Pivotal tc Server download page. These instructions require **tc Server 2.9** or later.
2. The HTTP Session Management Module for tc Server is included in the VMware GemFire installation package. After you install VMware GemFire, you will find the module in the `tools/Modules` directory of the installation with a name of the form `Apache_Geode_Modules-SERVER-VERSION-tcServer30.zip` or `Apache_Geode_Modules-SERVER-VERSION-tcServer.zip`. In these names,
   - SERVER-VERSION is the VMware GemFire version number.

  - `tcServer30` supports tcServer version 3.x.
  - `tcserver` supports tcServer version 2.9.

3. Unzip the appropriate module into the Pivotal tc Server `$CATALINA_HOME/templates` directory so that it creates `geode-p2p` and `geode-cs` subdirectories within the tc Server `templates` directory.

Setting Up the HTTP Module for tc Server

To set up the HTTP Module for tc Server, start a tc Server instance with the appropriate tc Server template based on your preferred topology.

Note: In the procedures that follow, you may be required to log in as root or use sudo to run the commands in Unix, especially if you installed Pivotal tc Server from RPM using yum.

Setup and Start

Edit the configuration in the appropriate file for the topology to set the locator host and port. For peer-to-peer topology, edit `geode-p2p/conf/catalina.properties` to set the `geode-p2p.locators` property. For client/server topology, edit `geode-cs/conf/cache-client.xml`.

Create a tc Server instance using one of these commands, choosing the appropriate template for the topology. Add the `--version` option if you need to pin your tc Server instance to a specific version of the tc Runtime:

```
In Unix, for a peer-to-peer topology:
  $ ./tcruntime-instance.sh create my_instance_name --template geode-p2p

In Unix, for a peer-to-peer topology, with a pinned tc Server instance:
  $ ./tcruntime-instance.sh create --version 6.0.32.A.RELEASE \
  my_instance_name --template geode-p2p

In Unix, for a client/server topology:
  $ ./tcruntime-instance.sh create my_instance_name --template geode-cs

In Unix, for a client/server topology, with a pinned tc Server instance:
  $ ./tcruntime-instance.sh create  --version 6.0.32.A.RELEASE \
  my_instance_name --template geode-cs

In Windows, for a peer-to-peer topology:
  $ tcruntime-instance.bat create my_instance_name --template geode-p2p

In Windows, for a peer-to-peer topology, with a pinned tc Server instance:
  $ tcruntime-instance.bat create --version 6.0.32.A.RELEASE ^
  my_instance_name --template geode-p2p

In Windows, for a client/server topology:
  $ tcruntime-instance.bat create my_instance_name --template geode-cs

In Windows, for a client/server topology, with a pinned tc Server instance:
  $ tcruntime-instance.bat create  --version 6.0.32.A.RELEASE ^
  my_instance_name --template geode-cs
```

To run, first start the locator, which will listen on its default of port number 10334:

```
$ gfsh start locator --name=locator1
```

With a similar environment to this example that is for a client/server set up,

```
TC_VER=tomcat-8.0.30.C.RELEASE
INSTANCE=geode-cs
CLASSPATH=$PWD/$INSTANCE/lib/geode-modules-1.0.0.jar:\
$PWD/$INSTANCE/lib/geode-modules-tomcat8-1.0.0.jar:\
$PWD/$TC_VER/lib/servlet-api.jar:\
$PWD/$TC_VER/lib/catalina.jar:\
$PWD/$TC_VER/lib/tomcat-util.jar:\
$PWD/$TC_VER/bin/tomcat-juli.jar
```

Start the server using `gfsh`:

```
$ gfsh start server --name=server1 --locators=localhost[10334] \
        --server-port=0 --classpath=$CLASSPATH
```

Starting the Application Server

Once you've created a tc Server instance, you are ready to start the instance.

```
In Unix:
  $ ./tcruntime-ctl.sh my_instance_name start

In Windows:
  $ ./tcruntime-ctl.bat my_instance_name start
```

Refer to the tc Server documentation for more information. Once started, VMware GemFire will automatically launch within the application server process.

Note: VMware GemFire session state management provides its own clustering functionality. If you are using VMware GemFire, you should NOT turn on Tomcat clustering as well.

To verify that the system is running, check the log file for a message similar to:

```
Mar 29, 2016 8:38:31 AM org.apache.geode.modules.session.bootstrap.AbstractCache
lifecycleEvent
INFO: Initializing VMware GemFire Modules
Modules version: 1.0.0
Java version:   1.0.0 user1 032916 2016-11-29 07:49:26 -0700
javac 1..0_272
Native version: native code unavailable
Source revision: c36591b73243c7ee3a0186710338453d12efe364
Source repository: develop
Running on: /192.0.2.0, 8 cpu(s), x86_64 Mac OS X 10.11.4
```

Information is also logged within the VMware GemFire log file, which by default is named `gemfire_modules.log`.

Changing the Default VMware GemFire Configuration in the tc Server Module

By default, the tc Server HTTP module will run VMware GemFire automatically with pre-configured settings. You can change these VMware GemFire settings.

Here are the default VMware GemFire settings:

- VMware GemFire peer-to-peer members use locators for discovery.
- The region name is set to `gemfire_modules_sessions`.
- The cache region is replicated for peer-to-peer configurations and partitioned (with redundancy turned on) for client/server configurations.
- VMware GemFire clients have local caching turned on and when the local cache needs to evict data, it will evict least-recently-used (LRU) data first.

Note: On the application server side, the default inactive interval for session expiration is set to 30 minutes. To change this value, refer to Session Expiration.

To change this default configuration, use the `--interactive` command line argument when creating the tc Server instance. Choose the appropriate command for the chosen topology and platform.

```
In Unix, for a peer-to-peer topology:
  $ ./tcruntime-instance.sh create my_instance_name --template gemfire-p2p \
        --interactive

In Unix, for a client/server topology:
  $ ./tcruntime-instance.sh create my_instance_name --template gemfire-cs \
         --interactive

In Windows, for a peer-to-peer topology:
  $ tcruntime-instance.bat create my_instance_name --template gemfire-p2p ^
         --interactive

In Windows, for a client/server topology:
  $ tcruntime-instance.bat create my_instance_name --template gemfire-cs ^
         --interactive
```

In interactive mode, you will be prompted to specify a series of property values. Hit <return> for any property that should use the default value.

After responding to approximately 20 prompts, you should see the following line:

```
Instance created.
```

Note: You cannot override region attributes on the cache server when using the HTTP Session Management Module. You must place all region attribute definitions in the region attributes template that you customize in tc Server. See Overriding Region Attributes for more information.

For information on setting up your instance for the most common types of configurations, refer to the sections below. For more information about each interactive prompt, refer to Interactive Mode Reference.

Using a Different Locator Port

For a VMware GemFire peer-to-peer member to communicate on a different port than the default (10334), answer the following question in the tc Server HTTP module's interactive mode:

```
Please enter the list of locators used by GemFire members to discover each other.
The format is a comma-separated list of host[port]. Default 'localhost[10334]': localh
ost[10335]
```

This example changes the locator port to 10335 on localhost.

Overriding Region Attributes

When using the HTTP Session Management Module, you cannot override region attributes directly on the cache server. You must place all region attribute definitions in the region attributes template that you customize within tc Server. For example, to specify a different name for the region's disk store, you could add the new disk-store-name specification to the region attributes template and then reference the template on the cache server.

```
<region-attributes id="MY_SESSIONS" refid="PARTITION_REDUNDANT_PERSISTENT_OVERFLOW"
disk-store-name="mystore">
  ...
</region-attributes>
```

Then on the cache server side, reference the modified region attributes template to allow the region to use the disk-store-name attribute:

```
<region name="gemfire_modules_sessions" refid="MY_SESSIONS"/>
```

- **Interactive Configuration Reference for the tc Server Module**

  This section describes each prompt when entering into interactive configuration mode of the VMware GemFire HTTP Session Management Module for tc Server.

Interactive Configuration Reference for the tc Server Module

This section describes each prompt when entering into interactive configuration mode of the VMware GemFire HTTP Session Management Module for tc Server.

```
  Please enter a value for 'geode-cs.maximum.vm.heap.size.mb'. Default '512':
  Please enter a value for 'geode-cs.initial.vm.heap.size.mb'. Default '512':
  Please enter a value for 'geode-cs.cms.initiating.heap.percentage'. Default '50':
```

The above properties allow you to fine-tune your JVM heap and garbage collector. For more information, refer to Managing Heap and Off-heap Memory.

```
  Please specify whether to enable a VMware GemFire listener that logs session create,
  update, destroy and expiration events. Default 'false':
```

The above property determines whether a debug cache listener is added to the session region. When true, info-level messages are logged to the VMware GemFire log when sessions are created, updated, invalidated, or expired.

```
With the geode-p2p template:
  Please specify whether to maintain a local VMware GemFire cache. Default 'false':

With the geode-cs template:
  Please specify whether to maintain a local VMware GemFire cache. Default 'true':
```

The above property determines whether a local cache is enabled; if this parameter is set to true, the app server load balancer should be configured for sticky session mode.

```
With the geode-p2p template:
  Please enter the id of the attributes of the VMware GemFire region used to cache
      sessions.
  Default 'REPLICATE':

With the geode-cs template:
  Please enter the id of the attributes of the VMware GemFire region used to cache
      sessions.
  Default 'PARTITION_REDUNDANT':
```

The above property determines the ID of the attributes for the cache region; possible values include PARTITION, PARTITION_REDUNDANT, PARTITION_PERSISTENT, REPLICATE, REPLICATE_PERSISTENT, and any other region shortcut that can be found in Region Shortcuts and Custom Named Region Attributes. When using a partitioned region attribute, it is recommended that you use PARTITION_REDUNDANT (rather than PARTITION) to ensure that the failure of a server does not result in lost session data.

```
  Please enter the name of the VMware GemFire region used to cache sessions.
  Default 'gemfire_modules_sessions':
```

The above property determines the VMware GemFire region name.

```
  Please enter the port that Tomcat Shutdown should listen on. Default '-1':
  Please enter the port that the JMX socket listener should listen on.
      Default '6969':
```

The above properties are application server properties.

```
  Please enter a value for 'bio.http.port'. Default '8080':
  Please enter a value for 'bio.https.port'. Default '8443':
```

tc Server requires information about connector ports. `bio.http.port` is the http port for tc Server and `bio.https.port` is the secure http port for tc Server.

```
With the geode-p2p template:
  Please enter the name of the VMware GemFire cache configuration file.
  Default 'cache-peer.xml':

With the geode-cs template:
  Please enter the name of the VMware GemFire cache configuration file.
  Default 'cache-client.xml':
```

You can change the name of the cache configuration file with the above property. If you do change this value, be sure to include an xml file by that name in the `conf` subdirectory.

```
  Please enter the percentage of heap at which updates to the cache are refused.
  0.0 means disabled. Default '0.0':
  Please enter the percentage of heap at which sessions will be evicted from the
  local cache. Default '80.0':
```

The above properties allow you to control the critical and eviction watermarks for the heap. By default, the critical watermark is disabled (set to 0.0) and the eviction watermark is set to 80%.

```
Please enter the list of locators used by VMware GemFire members to discover each othe
r.
The format is a comma-separated list of host[port]. Default ' ':
```

The above property specifies the list of locators.

```
  Please enter the name of the file used to log VMware GemFire messages.
  Default 'gemfire_modules.log':
```

The above property determines the file name for the VMware GemFire log file.

```
Applicable to the geode-p2p template ONLY:
  Please specify whether to rebalance the VMware GemFire cache at startup.
  Default 'false':
```

This property allows you to rebalance a partitioned VMware GemFire cache when a new VMware GemFire peer starts up.

```
  Please enter the name of the file used to store VMware GemFire statistics.
  Default 'gemfire_modules.gfs':
```

The above property determines the filename for the VMware GemFire statistics file.

```
  Please specify whether VMware GemFire statistic sampling should be enabled.
  Default 'false':
```

The above property determines whether statistics sampling should occur. See Statistics for more information.

HTTP Session Management Module for Tomcat

You set up and use the module by modifying the Tomcat's `server.xml` and `context.xml` files.

For instructions specific to SpringSource tc Server templates, refer to HTTP Session Management Module for Pivotal tc Server.

- **Installing the HTTP Module for Tomcat**

  This topic describes how to install the HTTP session management module for Tomcat.

- **Setting Up the HTTP Module for Tomcat**

  To use the VMware GemFire HTTP module with Tomcat application servers, you will need to modify Tomcat's `server.xml` and `context.xml` files.

- **Changing the Default VMware GemFire Configuration in the Tomcat Module**

By default, the Tomcat module will run VMware GemFire automatically with pre-configured settings. You can change these VMware GemFire settings.

Installing the HTTP Module for Tomcat

This topic describes how to install the HTTP session management module for Tomcat.

1. If you have not already installed Tomcat, download the desired version from the Apache Website and install it.
2. Following the Apache Tomcat convention, this page assumes the CATALINA_HOME environment variable is set to the root directory of the “binary” Tomcat distribution. For example, if Apache Tomcat is installed in `/usr/bin/apache-tomcat-9.0.62` then

   ```
   CATALINA_HOME=/usr/bin/apache-tomcat-9.0.62
   ```

   Define $CATALINA_HOME if it is not already defined.
3. For development purposes, you may find it helpful to establish manager-level access to Tomcat. To do so, edit the file `$CATALINA_HOME/conf/tomcat-users.xml`. Uncomment the following line, and replace the `<must-be-changed>` placeholder with a password of your own choosing:

   ```
     <user username="admin" password="<must-be-changed>" roles="manager-gui"/>
   ```

   This creates an admin/password credential you can use to view system information when following links from the Tomcat home page.
4. The HTTP Session Management Module for Tomcat is included in the VMware GemFire installation package. After you install VMware GemFire, you will find the module in the `tools/Modules` directory of the installation with a name of the form `Apache_Geode_Modules-SERVER-VERSION-Tomcat.zip`, where SERVER-VERSION is the VMware GemFire version number.
5. Set your current working directory to the `$CATALINA_HOME` directory (or wherever you installed the application server) and unzip the HTTP Session Management Module. This adds jar files to the `lib` subdirectory and XML files to the `conf` subdirectory.

   ```
   cd $CATALINA_HOME
   unzip $GEODE_HOME/tools/Modules/Apache_Geode_Modules-SERVER-VERSION-Tomcat.zip
   ```
6. Copy all of the jar files from the VMware GemFire `lib` subdirectory to the `lib` subdirectory of your Tomcat server (`$CATALINA_HOME/lib`):

   ```
   cd $CATALINA_HOME/lib
   cp $GEODE_HOME/lib/*.jar .
   ```

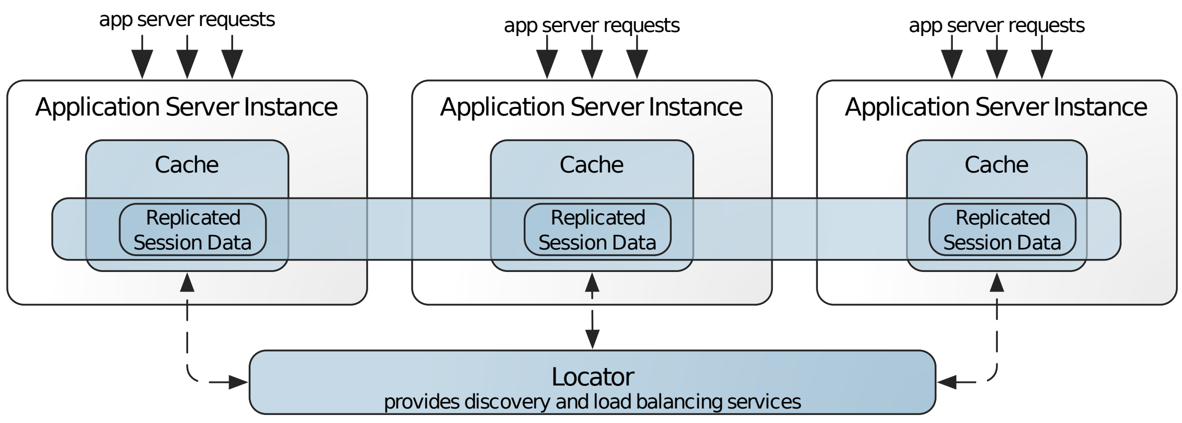
Proceed to Setting Up the HTTP Module for Tomcat to complete your Tomcat configuration.

Setting Up the HTTP Module for Tomcat

To use the VMware GemFire HTTP module with Tomcat application servers, you will need to modify Tomcat's `server.xml` and `context.xml` files.

Configuration is slightly different depending on the topology you are setting up: peer-to-peer or client/server. Refer to Common Topologies for HTTP Session Management for more information.

Peer-to-Peer Setup



To run VMware GemFire in a peer-to-peer configuration, add the following line to Tomcat's `$CATALINA_HOME$/conf/server.xml` within the `<Server>` tag:

```
<Listener className="org.apache.geode.modules.session.catalina.PeerToPeerCacheLifecycl
eListener"
        locators="localhost[10334]" />
```

Depending on the version of Tomcat you are using, add one of the following lines to `$CATALINA_HOME$/conf/context.xml` within the `<Context>` tag:

For Tomcat 7.0:

```
<Manager className="org.apache.geode.modules.session.Tomcat7DeltaSessionManager"/>
```

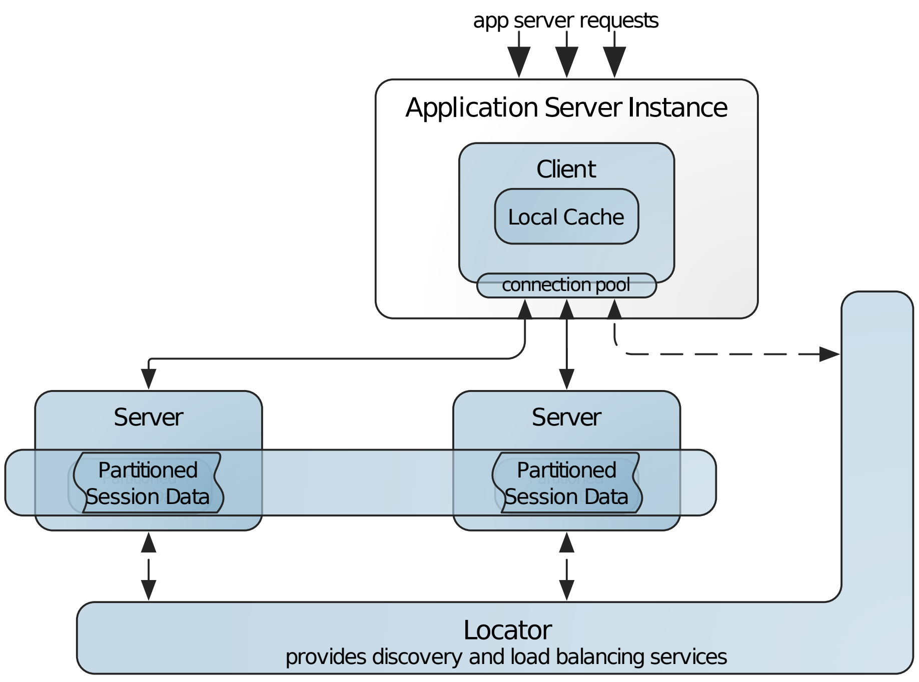
For Tomcat 8.0 and 8.5:

```
<Manager className="org.apache.geode.modules.session.catalina.Tomcat8DeltaSessionManag
er"/>
```

For Tomcat 9.0:

```
<Manager className="org.apache.geode.modules.session.catalina.Tomcat9DeltaSessionManag
er"/>
```

Client/Server Setup



To run VMware GemFire in a client/server configuration, the application server will operate as a VMware GemFire client. To do this, add the following line to `$CATALINA_HOME$/conf/server.xml` within the `<Server>` tag:

```
<Listener className="org.apache.geode.modules.session.catalina.ClientServerCacheLifecy
cleListener"/>
```

Depending on the version of Tomcat you are using, add one of the following lines to `$CATALINA_HOME$/conf/context.xml` within the `<Context>` tag:

For Tomcat 7.0:

```
<Manager className="org.apache.geode.modules.session.catalina.Tomcat7DeltaSessionManag
er"/>
```

For Tomcat 8.0 and 8.5:

```
<Manager className="org.apache.geode.modules.session.catalina.Tomcat8DeltaSessionManag
er"/>
```

For Tomcat 9.0:

```
<Manager className="org.apache.geode.modules.session.catalina.Tomcat9DeltaSessionManag
er"/>
```

The application server operates as a VMware GemFire client in this configuration.

Set the CLASSPATH environment variable. For a client/server set up using Apache Tomcat v9 and Geode v1.13, the CLASSPATH setting should be similar to the following. Adjust filenames and version numbers as needed for your implementation.

```
CLASSPATH="$CATALINA_HOME/lib/geode-modules-1.13.3.jar:\
$CATALINA_HOME/lib/geode-modules-tomcat9-1.13.3.jar:\
$CATALINA_HOME/lib/servlet-api.jar:\
$CATALINA_HOME/lib/catalina.jar:\
$CATALINA_HOME/lib/tomcat-util.jar:\
$CATALINA_HOME/bin/tomcat-juli.jar"
```

Start the locator and server using `gfsh`:

```
$ gfsh start locator --name=locator1 --classpath=$CLASSPATH
$ gfsh start server --name=server1 --locators=localhost[10334] --server-port=0 \
  --classpath=$CLASSPATH
```

Starting the Application Server

Once you've updated the configuration, you are now ready to start your tc Server or Tomcat instance. Refer to your application server documentation for starting the application server. Once started, VMware GemFire will automatically launch within the application server process.

Note: VMware GemFire session state management provides its own clustering functionality. If you are using VMware GemFire, you should NOT turn on Tomcat clustering as well.

Verifying that VMware GemFire Started

You can verify that VMware GemFire has successfully started by inspecting the Tomcat log file. For example:

```
15-Jul-2021 10:25:11.483 INFO [main] org.apache.catalina.startup.HostConfig.deployDire
ctory Deployment of web application directory [/Users/user/workspace/apache-tomcat-9.
0.50/webapps/host-manager] has finished in [1,688] ms
15-Jul-2021 10:25:11.486 INFO [main] org.apache.coyote.AbstractProtocol.start Starting
ProtocolHandler ["http-nio-8080"]
15-Jul-2021 10:25:11.493 INFO [main] org.apache.catalina.startup.Catalina.start Server
startup in [11682] milliseconds
```

Changing the Default GemFire Configuration in the Tomcat Module

By default, the Tomcat module will run VMware GemFire automatically with pre-configured settings. You can change these VMware GemFire settings.

Here are the default settings:

- Locators are used for member discovery.
- The region name is set to `gemfire_modules_sessions`.
- The cache region is replicated for peer-to-peer configurations and partitioned (with redundancy turned on) for client/server configurations.
- VMware GemFire clients have local caching turned on and when the local cache needs to evict data, it will evict least-recently-used (LRU) data first.

Note: On the application server side, the default inactive interval for session expiration is set to 30 minutes. For information about changing this value, see Session Expiration in *General Information on HTTP Session Management*.

Changing VMware GemFire Distributed System Properties

VMware GemFire system properties must be set by adding properties to Tomcat's `server.xml` file. When setting properties, use the following syntax:

```
<Listener
  className="org.apache.geode.modules.session.catalina.xxxLifecycleListener"
  property-name="property-value"
  property-name="property-value"
  ...
/>
```

Where:

- `property-name` is the name of a property.
- `property-value` is value of that property.

If the `xxxLifecycleListener` is a `PeerToPeerCacheLifecycleListener`, then a minimal addition to the `server.xml` file is the following:

```
<Listener
  className="org.apache.geode.modules.session.catalina.
     PeerToPeerCacheLifecycleListener"
  cache-xml-file="cache-peer.xml"
  locators="localhost[10334]"
 />
```

The list of Tomcat's configurable `server.xml` system properties includes any of the properties that can be specified in VMware GemFire's `gemfire.properties` file. The following list contains some of the properties that can be configured.

Property	Description	Default
cache-xml-file	Name of the cache configuration file	`cache-peer.xml` for peer-to-peer, `cache-client.xml` for client/server
locators (only for peer-to-peer topology)	**Required**: A list of locators in `(host\[port\])` format used by VMware GemFire members. If a single locator listens on its default port, set this value to `"localhost[10334]"`	Empty string
log-file	Name of the VMware GemFire log file	`gemfire_modules.log`
statistic-archive-file	Name of the VMware GemFire statistics file	`gemfire_modules.gfs`
statistic-sampling-enabled	Whether VMware GemFire statistics sampling is enabled	false

For more information about these properties, along with the full list of properties, see VMware GemFire Reference.

In addition to the standard VMware GemFire system properties, the following cache-specific properties can also be configured with the `LifecycleListener`.

Property	Description	Default
criticalHeapPercentage	Percentage of heap at which updates to the cache are refused	0 (Deactivated)
evictionHeapPercentage	Percentage of heap at which session eviction begins	80.0
rebalance	Whether a rebalance of the cache should be done when the application server instance is started	false

Although these properties are not part of the standard VMware GemFire system properties, they apply to the entire JVM instance and are therefore also handled by the `LifecycleListener`. For more information about managing the heap, see Controlling Heap Use with the Resource Manager in *Managing Heap Memory*.

Changing Cache Configuration Properties

To edit VMware GemFire cache properties such as the name and the characteristics of the cache region, add these properties to Tomcat's `context.xml` file. When adding properties, unless otherwise specified, use the following syntax:

```
<Manager
  className="org.apache.geode.modules.session.catalina.Tomcat9DeltaSessionManager"
  property-name="property-value"
  property-name="property-value"
  ...
/>
```

Where:

- `property-name` is the name of a property.
- `property-value` is the value of that property.

For example, this entry creates a partitioned region with the name **my_region**.

```
<Manager className="org.apache.geode.modules.session.catalina.
      Tomcat9DeltaSessionManager"
  regionAttributesId="PARTITION_REDUNDANT"
  regionName="my_region"
/>
```

The following parameters are the cache configuration parameters that can be added to Tomcat's `context.xml` file.

CommitSessionValve

Whether to wait until the end of the HTTP request to save all session attribute changes to the VMware GemFire cache. If the configuration line is present in the application's `context.xml` file, then only one put will be performed into the cache for the session per HTTP request. If the configuration line is not included, then the session is saved each time the `setAttribute` or `removeAttribute` method is invoked. As a consequence, multiple puts are performed into the cache during a single session. This configuration setting is recommended for any applications that modify the session frequently during a single HTTP request.

Default: Set

To deactivate this configuration, remove or comment out the following line from Tomcat's `context.xml` file.

```
<Valve className="org.apache.geode.modules.session.catalina.CommitSessionValve"/>
```

---

enableDebugListener

Whether to enable a debug listener in the session region. If this parameter is set to true, info-level messages are logged to the GemFire log when sessions are created, updated, invalidated or expired.

Default: `false`

The VMware GemFire API equivalent to setting this parameter:

```
// Create factory
AttributesFactory factory = ...; <or> RegionFactory factory = ...;
// Add cache listener
factory.addCacheListener(new DebugCacheListener());
```

---

enableLocalCache

Whether a local cache is enabled. If this parameter is set to true, the app server load balancer should be configured for sticky session mode.

Default: `true` for client/server. **enableLocalCache** is only useful with a client/server topology.

The VMware GemFire API equivalent to setting this parameter:

```
ClientCache.createClientRegionFactory(CACHING_PROXY_HEAP_LRU);
```

---

regionAttributesId

Specifies the region shortcut. For more information, see Region Shortcuts and Custom Named Region Attributes. When using a partitioned region attribute, VMware recommends that you use `PARTITION_REDUNDANT` instead of `PARTITION` to ensure that the failure of a server does not result in lost session data.

Default: `REPLICATE` for peer-to-peer, `PARTITION_REDUNDANT` for client/server

The VMware GemFire API equivalent to setting this parameter:

```
// Creates a region factory for the specified region shortcut
Cache.createRegionFactory(regionAttributesId);
```

---

regionName

Name of the region.

Default: `gemfire_modules_sessions`

The VMware GemFire API equivalent to setting this parameter:

```
// Creates a region with the specified name
RegionFactory.create(regionName);
```

HTTP Session Management Module for AppServers

You implement session caching with the HTTP Session Management Module for AppServers with a special filter, defined in the `web.xml`, which is configured to intercept and wrap all requests.

You can use this HTTP module with a variety of application servers. Wrapping each request allows the interception of `getSession()` calls to be handled by VMware GemFire instead of the native container. This approach is a generic solution, which is supported by any container that implements the Servlet 3.1 specification.

- **Setting Up the HTTP Module for AppServers**

  To use the module, you need to modify your application's `web.xml` files. Configuration is slightly different depending on the topology you are setting up.

- **Changing the Default VMware GemFire Configuration in the AppServers Module**

  By default, the AppServers module will run VMware GemFire automatically with preconfigured settings. You can change these VMware GemFire settings.

Setting Up the HTTP Module for AppServers

To use the module, you need to modify your application's `web.xml` files. Configuration is slightly different depending on the topology you are setting up.

Refer to Common Topologies for HTTP Session Management for more information. Modifying the war file can be done manually or with the `modify_war` script. To see the command line options for the `modify_war` script, invoke:

```
$ modify_war -h
```

Manual Configuration

To modify your war or ear file manually, make the following updates:

- **web.xml** needs a filter added as follows. If you have your own filters, the VMware GemFire Module filter **must** be the first one.

  ```
  <filter>
      <filter-name>gemfire-session-filter</filter-name>
      <filter-class>
        org.apache.geode.modules.session.filter.SessionCachingFilter
      </filter-class>
      <init-param>
          <param-name>cache-type</param-name>
          <param-value>peer-to-peer</param-value>
      </init-param>
      <init-param>
          <param-name>gemfire.property.locators</param-name>
          <param-value>localhost[10334]</param-value>
      </init-param>
  ```

```
</filter>
<filter-mapping>
    <filter-name>gemfire-session-filter</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
```

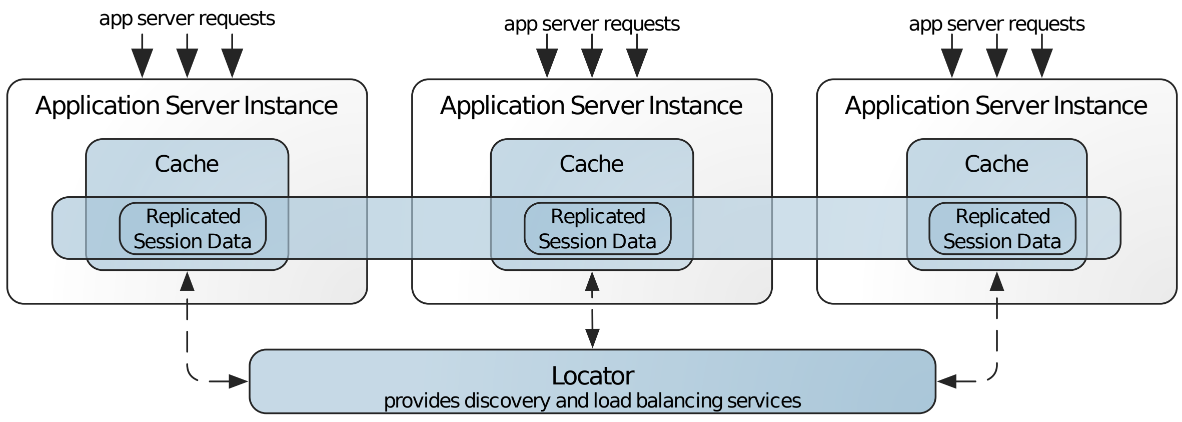
- Add the following jar files from the AppServer.zip to the `WEB-INF/lib` directory of the war:
  - geode-modules jar
  - geode-modules-session-internal jar
  - geode-modules-session jar
  - slf4j-api jar
  - slf4j-jdk14 jar
- Add the following jar files from the `$GEODE/lib` directory to the `WEB-INF/lib` directory of the war, where `$GEODE` is set to the VMware GemFire product installation:
  - antlr jar
  - commons-io jar
  - commons-lang jar
  - commons-validator jar
  - fastutil jar
  - geode-common jar
  - geode-core jar
  - geode-deployment-legacy jar
  - geode-json jar
  - geode-management jar
  - geode-logging jar
  - geode-serialization jar
  - javax.transaction-api jar
  - jgroups jar
  - log4j-api jar
  - log4j-core jar
  - log4j-jul jar
  - micrometer-core jar
  - shiro-core jar

If you are deploying an ear file:

- Copy all the dependent files, given above, to the `lib` directory of the ear.
- Modify each embedded war file's manifest by adding a Class-Path entry which references the shared jars added in the previous step. For example:

```
Manifest-Version: 1.0
Built-By: joe
Build-Jdk: 1.8.0_77
Created-By: Apache Maven
Archiver-Version: Plexus Archiver
Class-Path: lib/geode-modules-1.0.0.jar
lib/geode-modules-session-internal-1.0.0.jar
lib/geode-modules-session-1.0.0.jar
lib/slf4j-api-1.7.7.jar
lib/slf4j-jdk14-1.7.7.jar
lib/antlr-2.7.7.jar
lib/fastutil-7.0.2.jar
lib/geode-core-1.0.0.jar
lib/geode-common.1.0.0.jar
lib/geode-management.1.0.0.jar
lib/geode-logging.1.0.0.jar
lib/geode-serialization.1.0.0.jar
lib/javax.transaction-api-1.3.jar
lib/jgroups-3.6.8.Final.jar
lib/log4j-api-2.5.jar
lib/log4j-core-2.5.jar
lib/log4j-jul-2.5.jar
```

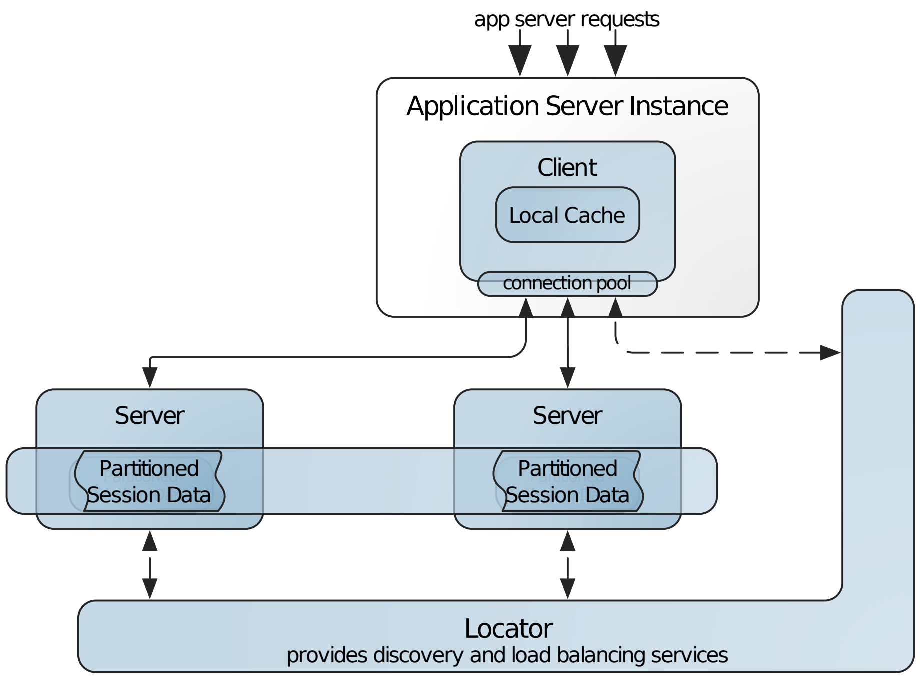
Peer-to-Peer Setup



To run VMware GemFire in a peer-to-peer configuration, use the `modify_war` script with options `-t peer-to-peer`, `-p gemfire.property.locators=localhost[10334]`, and `-p gemfire.propery.cache-xml-file=<moduleDir>/conf/cache-peer.xml` to result in the following `web.xml` content:

```
<filter>
    <filter-name>gemfire-session-filter</filter-name>
    <filter-class>
        org.apache.geode.modules.session.filter.SessionCachingFilter
    </filter-class>
    <init-param>
        <param-name>cache-type</param-name>
        <param-value>peer-to-peer</param-value>
    </init-param>
    <init-param>
        <param-name>gemfire.property.locators</param-name>
        <param-value>localhost[10334]</param-value>
    </init-param>
</filter>
```

Client/Server Setup



To run VMware GemFire in a client/server configuration, you make the application server operate as a VMware GemFire client. Use the `-t client-server` option to the `modify_war` script. This adds the following filter to application server's `web.xml` file:

To run VMware GemFire in a client/server configuration, you make the application server operate as a VMware GemFire client. Use the `modify_war` script with options `-t client-server` and `-p gemfire.property.cache-xml-file=<module dir>/conf/cache-client.xml` to result in the following `web.xml` content:

```
<filter>
    <filter-name>gemfire-session-filter</filter-name>
    <filter-class>
        org.apache.geode.modules.session.filter.SessionCachingFileter
    </filter-class>
    <init-param>
        <param-name>cache-type</param-name>
        <param-value>client-server</param-value>
    </init-param>
    <init-param>
        <param-name>gemfire.property.cache-xml-file</param-name>
        <param-value>module dir/conf/cache-client.xml</param-value>
    </init-param>
</filter>
```

The `cache-client.xml` file contains a <pool> element pointing at the locator. Its default value is localhost[10334].

Starting the Application Server

After you update the configuration, you are now ready to start your application server instance. Instantiate the locator first:

```
$ gfsh start locator --name=locator1
```

Then start the server:

```
$ gfsh start server \
    --name=server1 \
    --server-port=0 \
    --locators=localhost[10334] \
    --classpath=<moduleDir>/lib/geode-modules-1.0.0.jar:\
<moduleDir>/lib/geode-modules-session-internal-1.0.0.jar
```

Once the application server is started, the VMware GemFire client will automatically launch within the application server process.

Verifying that VMware GemFire Started

You can verify that VMware GemFire has successfully started by inspecting the application server log file. For example:

```
info 2016/04/18 10:04:18.685 PDT <localhost-startStop-2> tid=0x1a]
Initializing VMware GemFire Modules
Java version:   1.0.0 user1 041816 2016-11-18 08:46:17 -0700
javac 1..0_272
Native version: native code unavailable
Source revision: 19dd8eb1907e0beb2aa3e0a17d5f12c6cbec6968
Source repository: develop
Running on: /192.0.2.0, 8 cpu(s), x86_64 Mac OS X 10.11.4
```

Information is also logged within the VMware GemFire log file, which by default is named `gemfire_modules.<date>.log`.

Changing the Default VMware GemFire Configuration in the AppServers Module

By default, the AppServers module will run VMware GemFire automatically with preconfigured settings. You can change these VMware GemFire settings.

Here are the default settings:

- VMware GemFire peer-to-peer members are discovered using locators.
- The region name is set to `gemfire_modules_sessions`.
- The cache region is replicated for peer-to-peer configurations and partitioned (with redundancy turned on) for client/server configurations.
- VMware GemFire clients have local caching turned on and when the local cache needs to evict data, it will evict least-recently-used (LRU) data first.

Note: On the application server side, the default inactive interval for session expiration is set to 30 minutes. To change this value, refer to Session Expiration.

However, you may want to change this default configuration. For example, you might want to change the region from replicated to partitioned. This section describes how to change these configuration values.

Note: You cannot override region attributes on the cache server when using the HTTP Session Management Module. You must place all region attribute definitions in the region attributes template that you customize in your application server. See Overriding Region Attributes for more information.

Changing VMware GemFire Distributed System Properties

To edit VMware GemFire system properties, you must add properties to VMware GemFire Session Filter definition in the application's web.xml file. As mentioned previously, this can be done by using the **-p** option to the `modify_war` script. All VMware GemFire system properties should be prefixed with the string **gemfire.property**. For example:

- **-p gemfire.property.locators=hostname[10334]**
- **-p gemfire.property.cache-xml-file=/u01/weblogic/conf/cache.xml**.

```
<filter>
    <filter-name>gemfire-session-filter</filter-name>
    <filter-class>
      org.apache.geode.modules.session.filter.SessionCachingFilter
    </filter-class>
    <init-param>
        <param-name>cache-type</param-name>
        <param-value>client-server</param-value>
    </init-param>
    <init-param>
        <param-name>gemfire.property.locators</param-name>
        <param-value>hostname[10334]</param-value>
    </init-param>
    <init-param>
        <param-name>gemfire.property.cache-xml-file</param-name>
        <param-value>/u01/weblogic/conf/cache.xml</param-value>
    </init-param>
</filter>
```

This example specifies that the file name for VMware GemFire's cache XML configuration is `cache-peer.xml`.

The list of configurable `server.xml` system properties include any of the properties that can be specified in VMware GemFire's `gemfire.properties` file. The following list contains some of the more common parameters that can be configured.

Parameter	Description	Default
cache-xml-file	Name of the cache configuration file.	`cache-peer.xml` for peer-to-peer, `cache-client.xml` for client/server

Parameter	Description	Default
locators (only for peer-to-peer config)	(required) list of locators (host[port]) used by VMware GemFire members; if a single locator listens on its default port, then set this value to `"localhost[10334]"`	Empty string
log-file	Name of the VMware GemFire log file.	`gemfire_modules.log`
statistic-archive-file	Name of the VMware GemFire statistics file.	`gemfire_modules.gfs`
statistic-sampling-enabled	Whether VMware GemFire statistics sampling is enabled.	false

In addition to the standard VMware GemFire system properties, the following cache-specific properties can also be configured.

Parameter	Description	Default
criticalHeapPercentage	Percentage of heap at which updates to the cache are refused.	0 (disabled)
evictionHeapPercentage	Percentage of heap at which session eviction begins.	80.0
rebalance	Whether a rebalance of the cache should be done when the application server instance is started.	false

Although these properties are not part of the standard VMware GemFire system properties, they apply to the entire JVM instance. For more information about managing the heap, refer to Managing Heap and Off-heap Memory.

Note: It is important to note that the VMware GemFire cluster is a singleton within the entire application server JVM. As such it is important to ensure that different web applications, within the same container, set (or expect) the same cache configuration. When the application server starts, the first web application to start that uses VMware GemFire Session Caching will determine the overall configuration of the cluster, since it will trigger the creation of the cluster.

Changing Cache Configuration Properties

To edit VMware GemFire cache properties (such as the name and the characteristics of the cache region), you must configure these using a filter initialization parameter prefix of **gemfire.cache** with the `modify_war` script. For example:

-p gemfire.cache.region_name=custom_sessions

```
<filter>
    <filter-name>gemfire-session-filter</filter-name>
    <filter-class>
      org.apache.geode.modules.session.filter.SessionCachingFilter
    </filter-class>
    <init-param>
        <param-name>cache-type</param-name>
        <param-value>peer-to-peer</param-value>
    </init-param>
    <init-param>
        <param-name>gemfire.cache.region_name</param-name>
        <param-value>custom_sessions</param-value>
    </init-param>
</filter>
```

The following parameters are the cache configuration parameters that can be added to the filter definition as initialization parameters.

enable\_debug\_listener

Whether to enable a debug listener in the session region; if this parameter is set to true, info-level messages are logged to the VMware GemFire log when sessions are created, updated, invalidated or expired.

Default: `false`

The VMware GemFire API equivalent to setting this parameter:

```
// Create factory
AttributesFactory factory = ...;
<or> RegionFactory factory = ...;
// Add cache listener
factory.addCacheListener(new DebugCacheListener());
```

enable\_local\_cache

Whether a local cache is enabled; if this parameter is set to true, the app server load balancer should be configured for sticky session mode.

Default: `false` for peer-to-peer, `true` for client/server

The VMware GemFire API equivalent to setting this parameter:

```
// For peer-to-peer members:
Cache.createRegionFactory(REPLICATE_PROXY)
// For client members:
ClientCache.createClientRegionFactory(CACHING_PROXY_HEAP_LRU)
```

region\_attributes\_id

Specifies the region shortcut; for more information refer to [Region Shortcuts and Custom Named Region Attributes](../../basic_config/data_regions/region_shortcuts.html); when using a partitioned region attribute, it is recommended that you use PARTITION\_REDUNDANT (rather than PARTITION) to ensure that the failure of a server does not result in lost session data.

Default: REPLICATE for peer-to-peer, PARTITION_REDUNDANT for client/server

The VMware GemFire API equivalent to setting this parameter:

```
// Creates a region factory for the specified region shortcut
Cache.createRegionFactory(regionAttributesId);
```

region\_name

Name of the region.

Default: gemfire_modules_sessions

The VMware GemFire API equivalent to setting this parameter:

```
// Creates a region with the specified name
```

```
RegionFactory.create(regionName);
```

session_delta_policy
Replication policy for session attributes.

Default: delta_queued

Delta replication can be configured to occur immediately when HttpSession.setAttribute() is called (delta_immediate) or when the HTTP request has completed processing (delta_queued). If the latter mode is configured, all attribute updates for a particular request are 'batched' and multiple updates to the same attribute are collapsed. Depending on the number of attributes updates within a given request, delta_queued may provide a significant performance gain. For complete session attribute integrity across the cache, delta_immediate is recommended. Note that this option is specific to this module and there is no equivalent VMware GemFire API to enable it.

- Common VMware GemFire Configuration Changes for AppServers

Common VMware GemFire Configuration Changes for AppServers

Overriding Region Attributes

When using the HTTP Session Management Module, you cannot override region attributes directly on the cache server. You must place all region attribute definitions in the region attributes template that you customize within the application server. For example, to specify a different name for the region's disk store, you could add the new disk-store-name specification to the region attributes template and then reference the template on the cache server.

```
<region-attributes id="MY_SESSIONS" refid="PARTITION_REDUNDANT_PERSISTENT_OVERFLOW"
disk-store-name="mystore">
  ...
</region-attributes>
```

Then on the cache server side, reference the modified region attributes template to allow the region to use the disk-store-name attribute:

```
<region name="gemfire_modules_sessions" refid="MY_SESSIONS"/>
```

Next, you must specify the region attributes ID as a value for the `region_attributes_id` parameter in web.xml. For example, if you want to enable the `region-attributes` in the above example for a specific Web application, you would configure the Web application's web.xml in the following manner:

```
<filter>
  ...
    <init-param>
        <param-name>gemfire.cache.region_attributes_id</param-name>
        <param-value>MY_SESSIONS</param-value>
    </init-param>
  ...
</filter>
```

Apache Lucene® Integration

Apache Lucene® is a widely used Java full-text search engine. This section describes how VMware GemFire integrates with Apache Lucene. We assume that the reader is familiar with Apache Lucene's indexing and search functionalities.

The Apache Lucene integration:

- Enables users to create Lucene indexes on data stored in VMware GemFire
- Provides high availability of indexes using VMware GemFire's HA capabilities to store the indexes in memory
- Colocates indexes with data
- For persistent regions, persists Lucene indexes to disk
- Updates the indexes asynchronously to minimize impacting write latency
- Provides scalability by partitioning index data

For more details, see Javadocs for the classes and interfaces that implement Apache Lucene indexes and searches, including `LuceneService`, `LuceneSerializer`, `LuceneIndexFactory`, `LuceneQuery`, `LuceneQueryFactory`, `LuceneQueryProvider`, and `LuceneResultStruct`.

Using the Apache Lucene Integration

You can interact with Apache Lucene indexes through a Java API, through the `gfsh` command-line utility, or by means of the `cache.xml` configuration file.

Key Points

- Apache Lucene indexes are supported only on partitioned regions. Replicated region types are *not* supported.
- Lucene indexes reside on servers. You cannot create a Lucene index on a client.
- A Lucene index applies to only one region. Multiple indexes can be defined for a single region.
- Heterogeneous objects in a single region are supported.

Creating a Lucene Index

Note: Create the Lucene index **before** creating the region.

When you create a Lucene index, you must provide three pieces of information:

1. The name of the index you wish to create
2. The name of the region to be indexed and searched
3. The names of the fields you wish to index

You must specify at least one field to be indexed.

If the object value for the entries in the region comprises a primitive type value without a field name, then use `__REGION_VALUE_FIELD` to specify the field to be indexed. `__REGION_VALUE_FIELD` serves as the field name for entry values of all primitive types, including `String`, `Long`, `Integer`, `Float`, and `Double`.

Each field has a corresponding analyzer to extract terms from text. When no analyzer is specified, the `org.apache.lucene.analysis.standard.StandardAnalyzer` is used.

The index has an associated serializer that renders the indexed object as a Lucene document comprised of searchable fields. The default serializer is a simple one that handles top-level fields, but does not render collections or nested objects.

VMware GemFire supplies a built-in serializer, `FlatFormatSerializer()`, that handles collections and nested objects. See Using FlatFormatSerializer to Index Fields within Nested Objects for more information regarding Lucene indexes for nested objects.

As a third alternative, you can create your own serializer, which must implement the `LuceneSerializer` interface.

Creating a Lucene Index: Java API Example

The following example uses the Java API to create a Lucene index with two fields. No analyzers are specified, so the default analyzer handles both fields. No serializer is specified, so the default serializer is used.

```
// Get LuceneService
LuceneService luceneService = LuceneServiceProvider.get(cache);

// Create the index on fields with default analyzer
//  prior to creating the region
luceneService.createIndexFactory()
  .addField("name")
  .addField("zipcode")
  .create(indexName, regionName);

Region region = cache.createRegionFactory(RegionShortcut.PARTITION)
  .create(regionName);
```

Creating a Lucene Index: Gfsh Example

In gfsh, use the create lucene index command to create Lucene indexes.

The following example creates an index with two fields. The default analyzer handles both fields, and the default serializer is used.

```
gfsh>create lucene index --name=indexName --region=/orders --field=customer,tags
```

The next example creates an index, specifying a custom analyzer for the second field. “DEFAULT” in the first analyzer position specifies that the default analyzer will be used for the first field.

```
gfsh>create lucene index --name=indexName --region=/orders
  --field=customer,tags --analyzer=DEFAULT,org.apache.lucene.analysis.bg.BulgarianAnal
yzer
```

Creating a Lucene Index: XML Example

This XML configuration file specifies a Lucene index with three fields and three analyzers:

```
<cache
    xmlns="http://geode.apache.org/schema/cache"
    xmlns:lucene="http://geode.apache.org/schema/lucene"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://geode.apache.org/schema/cache
        http://geode.apache.org/schema/cache/cache-1.0.xsd
        http://geode.apache.org/schema/lucene
        http://geode.apache.org/schema/lucene/lucene-1.0.xsd"
    version="1.0">

    <region name="region" refid="PARTITION">
        <lucene:index name="myIndex">
          <lucene:field name="a"
                        analyzer="org.apache.lucene.analysis.core.KeywordAnalyzer"/>
          <lucene:field name="b"
                        analyzer="org.apache.lucene.analysis.core.SimpleAnalyzer"/>
          <lucene:field name="c"
                        analyzer="org.apache.lucene.analysis.standard.ClassicAnalyze
r"/>
          <lucene:field name="d" />
        </lucene:index>
    </region>
</cache>
```

Using FlatFormatSerializer to Index Fields within Nested Objects

VMware GemFire supplies a built-in serializer, `org.apache.geode.cache.lucene.FlatFormatSerializer` that renders collections and nested objects as searchable fields, which you can access using the syntax `fieldnameAtLevel1.fieldnameAtLevel2` for both indexing and querying.

For example, in the following data model, the Customer object contains both a Person object and a collection of Page objects. The Person object also contains a Page object.

```
public class Customer implements Serializable {
  private String name;
  private Collection<String> phoneNumbers;
  private Collection<Person> contacts;
  private Page[] myHomePages;
  ......
}
public class Person implements Serializable {
  private String name;
  private String email;
  private int revenue;
  private String address;
  private String[] phoneNumbers;
  private Page homepage;
  .......
}
public class Page implements Serializable {
  private int id; // search integer in int format
  private String title;
  private String content;
  ......
}
```

The `FlatFormatSerializer` creates one document for each parent object, adding an indexed field for each data field in a nested object, identified by its qualified name. Similarly, collections are flattened and treated as tokens in a single field. For example, the `FlatFormatSerializer` could convert a Customer object, with the structure described above, into a document containing fields such as `name`, `contacts.name`, and `contacts.homepage.title` based on the indexed fields specified at index creation. Each segment is a field name, not a field type, because a class (such as Customer) could have more than one field of the same type (such as Person).

The serializer creates and indexes the fields you specify when you request index creation. The example below demonstrates how to index the `name` field and the nested fields `contacts.name`, `contacts.email`, `contacts.address`, `contacts.homepage.title`.

```
// Get LuceneService
LuceneService luceneService = LuceneServiceProvider.get(cache);

// Create Index on fields, some are fields in nested objects:
luceneService.createIndexFactory().setLuceneSerializer(new FlatFormatSerializer())
      .addField("name")
      .addField("contacts.name")
      .addField("contacts.email")
      .addField("contacts.address")
      .addField("contacts.homepage.title")
      .create("customerIndex", "Customer");

// Create region
Region CustomerRegion = ((Cache)cache).createRegionFactory(shortcut).create("Custome
r");
```

The gfsh equivalent of the above Java code uses the `create lucene index` command, with options specifying the index name, region name, field names, and the `FlatFormatSerializer`, specified using its fully qualified name,`org.apache.geode.cache.lucene.FlatFormatSerializer`:

```
gfsh>create lucene index --name=customerIndex --region=Customer
  --field=name,contacts.name,contacts.email,contacts.address,contacts.homepage.title
  --serializer=org.apache.geode.cache.lucene.FlatFormatSerializer
```

The syntax for querying a nested field is the same as for a top level field, but with the additional qualifying parent field name, such as `contacts.name:Jones77*`. This distinguishes which “name” field is intended when there can be more than one “name” field at different hierarchical levels in the object.

Java query:

```
LuceneQuery query = luceneService.createLuceneQueryFactory()
    .create("customerIndex", "Customer", "contacts.name:Jones77*", "name");

PageableLuceneQueryResults<K,Object> results = query.findPages();
```

gfsh query:

```
gfsh>search lucene --name=customerIndex --region=Customer
  --queryString="contacts.name:Jones77*"
  --defaultField=name
```

Queries

Querying a Lucene Index: Gfsh Example

For details, see the gfsh search lucene command reference page.

```
gfsh>search lucene --name=indexName --region=/orders --queryString="Jones*"
   --defaultField=customer
```

Querying a Lucene Index: Java API Example

```
LuceneQuery<String, Person> query = luceneService.createLuceneQueryFactory()
  .create(indexName, regionName, "name:John AND zipcode:97006", defaultField);

Collection<Person> results = query.findValues();
```

Destroying an Index

Since a region-destroy operation does not cause the destruction of any Lucene indexes, destroy any Lucene indexes prior to destroying the associated region.

Destroying a Lucene Index: Java API Example

```
luceneService.destroyIndex(indexName, regionName);
```

An attempt to destroy a region with a Lucene index will result in an `IllegalStateException`, issuing an error message similar to:

```
java.lang.IllegalStateException: The parent region [/orders] in colocation chain
 cannot be destroyed, unless all its children [[/indexName#_orders.files]] are
 destroyed
...
```

Destroying a Lucene Index: Gfsh Example

For details, see the gfsh destroy lucene index command reference page.

The error message that results from an attempt to destroy a region prior to destroying its associated Lucene index will be similar to:

```
Region /orders cannot be destroyed because it defines Lucene index(es)
  [/ordersIndex]. Destroy all Lucene indexes before destroying the region.
```

Changing an Index

Changing an index requires rebuilding it. Implement these steps to change an index:

1. Export all region data.
2. Destroy the Lucene index.
3. Destroy the region.
4. Create a new index.
5. Create a new region without the user-defined business logic callbacks.
6. Import the region data with the option to turn on callbacks. The callbacks will be to invoke a Lucene async event listener to index the data. The `gfsh import data` command will be of the form:

   ```
   gfsh>import data --region=myReg --member=M3 --file=myReg.gfd --invoke-callbacks
   =true
   ```

If the API is used to import data, the code to set the option to invoke callbacks will be similar to this code fragment:

~~~
``` pre
Region region = ...;
File snapshotFile = ...;
RegionSnapshotService service = region.getSnapshotService();
SnapshotOptions options = service.createOptions();
options.invokeCallbacks(true);
service.load(snapshotFile, SnapshotFormat.GEMFIRE, options);
```
~~~

1. Alter the region to add the user-defined business logic callbacks.

Additional Gfsh Commands

See the gfsh describe lucene index command reference page for the command that prints details about a specific index.

See the gfsh list lucene index command reference page for the command that prints details about the Lucene indexes created for all members.

Requirements and Caveats

- Join queries between regions are not supported.
- Lucene indexes are stored in on-heap memory only.
- Lucene queries from within transactions are not supported. On an attempt to query from within a transaction, a `LuceneQueryException` is thrown, issuing an error message on the client (accessor) similar to:

```
Exception in thread "main" org.apache.geode.cache.lucene.LuceneQueryException:
 Lucene Query cannot be executed within a transaction
...
```

- Lucene indexes must be created prior to creating the region. If an attempt is made to create a Lucene index after creating the region, the error message is similar to:

```
       Member                  | Status
---------------------------- | ------------------------------------------------------
192.0.2.0(s2:97639)<v2>:1026 | Failed: The lucene index must be created before region
192.0.2.0(s3:97652)<v3>:1027 | Failed: The lucene index must be created before region
192.0.2.0(s1:97626)<v1>:1025 | Failed: The lucene index must be created before region
```

- The order of server creation with respect to index and region creation is important. The cluster configuration service cannot work if servers are created after index creation, but before region creation, as Lucene indexes are propagated to the cluster configuration after region creation. To start servers at multiple points within the start-up process, use this ordering:
    1. start server(s)
    2. create Lucene index
    3. create region
    4. start additional server(s)
- An invalidate operation on a region entry does not invalidate a corresponding Lucene index entry. A query on a Lucene index that contains values that have been invalidated can return results that no longer exist. Therefore, do not combine entry invalidation with queries on Lucene indexes.
- Lucene indexes are not supported for regions that have eviction configured with a local destroy. Eviction can be configured with overflow to disk, but only the region data is overflowed to disk, not the Lucene index. On an attempt to create a region with eviction configured to do local destroy (with a Lucene index), an `UnsupportedOperationException` is thrown, issuing an error message similar to:

```
[error 2017/05/02 16:12:32.461 PDT <main> tid=0x1]
 java.lang.UnsupportedOperationException:
Exception in thread "main" java.lang.UnsupportedOperationException:
 Lucene indexes on regions with eviction and action local destroy are not supported
...
```

- Be aware that using the same field name in different objects where the field has different data types may have unexpected consequences. For example, if an index on the field SSN has the following entries
    - `Object_1 object_1` has String SSN = “1111”
    - `Object_2 object_2` has Integer SSN = 1111
    - `Object_3 object_3` has Float SSN = 1111.0

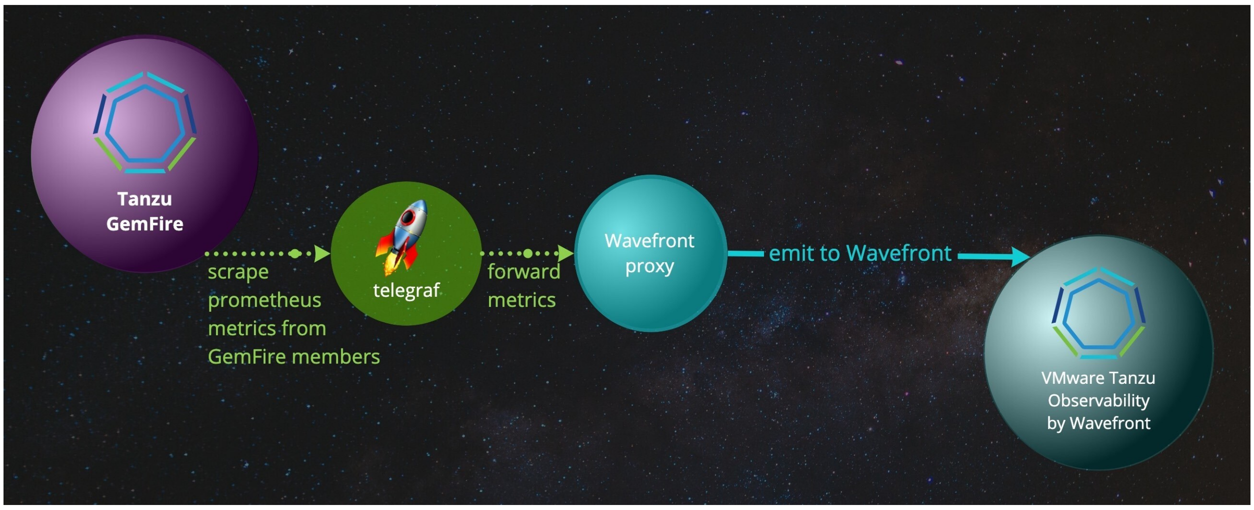
  Integers and floats will not be converted into strings. They remain as `IntPoint` and `FloatPoint` within Lucene. The standard analyzer will not try to tokenize these values. The standard analyzer will only try to break up string values. So, a string search for “SSN: 1111” will return `object_1`. An `IntRangeQuery` for `upper limit : 1112` and `lower limit : 1110` will return `object_2`, and a `FloatRangeQuery` with `upper limit : 1111.5` and `lower limit : 1111.0` will return `object_3`.

- Backups should only be made for regions with Lucene indexes when there are no puts, updates, or deletes in progress. A backup might cause an inconsistency between region data and a Lucene index. Both the region operation and the associated index operation cause disk operations, yet those disk operations are not done atomically. Therefore, if a backup were taken between the persisted write to a region and the resulting persisted write to the Lucene index, then the backup represents inconsistent data in the region and Lucene index.

Tanzu Observability by Wavefront

The VMware GemFire metrics module provides out-of-the-box integration with Tanzu Observability by Wavefront. The metrics module, provided as a collection of JAR files in your product release, adds a metrics endpoint to a specified VMware GemFire member. By default, this Prometheus-style endpoint hosts approximately 200 GemFire metrics at an update interval of 1 minute. These metrics can be scraped by a metrics collection agent (such as Telegraf) and forwarded to a metrics monitoring platform (such as Wavefront) for further analysis and alerting.

Delivering GemFire Metrics via Telegraf and Wavefront Proxy to Tanzu Observability by Wavefront


Tanzu GemFire
scrape prometheus metrics from GemFire members
telegraf
forward metrics
Wavefront proxy
emit to Wavefront
VMware Tanzu Observability by Wavefront

To enable Wavefront-viewable metrics across your VMware GemFire cluster:

1. If you wish to modify the default settings for number of emitted metrics or their refresh rate, do so by setting a system-wide environment variable.
2. For each member you wish to monitor, enable Wavefront-viewable Prometheus-style metrics when you create the member.

Configure GemFire Metrics

By default, each metrics endpoint hosts approximately 200 metrics at an update interval of one minute.

You can optionally configure the emission dataset and its update interval using the `GEMFIRE_METRICS` environment variable, which defines the `emission` and `interval` parameters using JSON syntax.

Setting `GEMFIRE_METRICS` is optional, but if you choose to do it, the configuration must be in place before you start the member to which it applies.

The `emission` parameter specifies the quantity of data to export:

Syntax	Value	Example
"emission" : "*value*"	**Default**: Emit approximately 200 metrics **All**: Emit all GemFire metrics **None**: No metrics will be available on the endpoint	"emission" : "All"

The `interval` property is used in computing certain windowed statistics. To get the most out of windowed statistics, align this interval to be close to your scrape interval. When set, this value overrides the default interval of one minute. An accepted value is a positive integer followed by a unit specifier:

- **s** for seconds
- **m** for minutes
- **h** for hours
- **d** for days
- **w** for weeks

For example, `4m` is four minutes, and `90s` is 90 seconds.

Note: When `emission` is set to `All`, the default for `interval` becomes `2s`, unless you specify otherwise.

Syntax	Value	Units	Examples
"interval" : "*value*"	positive integer followed by a unit	**s** seconds **m** minutes **d** days **w** weeks	"interval" : "4m" "interval" : "90s"

Sample usage:

```
export GEMFIRE_METRICS='{"emission": "All", "interval":"90s"}'
```

Enable Wavefront-Viewable Metrics

To enable Wavefront-viewable Prometheus-style metrics for a member (a VMware GemFire locator or server), you must provide the following information when you create the member:

- **Metrics JAR files**: Add the metrics JAR files to the classpath. The JAR files, located in the `gemfire-prometheus-metrics` directory, are included in your VMware GemFire distribution's `tools/Modules` directory. For example, if your product distribution is located in `/gemfire`, use the option `--classpath=/gemfire/tools/Modules/gemfire-prometheus-metrics/*` in the gfsh `start locator` or `start server` command.
- **Metrics port**: Specify a unique metrics port mapping for metrics collection. The metrics port is specified by Java command-line parameter `gemfire.prometheus.metrics.port`. In your gfsh `start` command, use the `--J=-D<param>=<value>` option to specify the parameter and its value. For example, `--J=-Dgemfire.prometheus.metrics.port=7001`.

1. To enable Wavefront-viewable metrics for a locator by adding the metrics JAR files to the classpath and specifying a metrics port, run the following `gfsh` command. If the `GEMFIRE_METRICS` environment variable is set, the metrics endpoint incorporates it into the member configuration.

   ```
   gfsh -e "start locator --classpath=/gemfire/tools/Modules/gemfire-prometheus-me
   trics/* \
   --J=-Dgemfire.prometheus.metrics.port=7001"
   ```

2. After the member has started, locate and record the IP address of the member in the output of the `start` command. For example, in the sample output below, the IP address is `192.168.50.245`:

   ```
   gfsh>start locator --classpath=/gemfire/tools/Modules/gemfire-prometheus-metric
   s/*
    --J=-Dgemfire.prometheus.metrics.port=7001
   Starting a VMware GemFire Locator in /Users/username/my_gemfire/locator1...
   .................
   Locator in /Users/username/my_gemfire/locator1 on 192.168.50.245[10334] as loca
   tor1 is currently
   online.
   Process ID: 4580
   Uptime: 10 seconds
   VMware GemFire Version: 9.10
   Java Version: 1..0_272
   Log File: /Users/username/my_gemfire/locator1/locator1.log
   JVM Arguments: -Dgemfire.enable-cluster-configuration=true
   -Dgemfire.load-cluster-configuration-from-dir=false
   -Dgemfire.launcher.registerSignalHandlers=true -Djava.awt.headless=true
   -Dsun.rmi.dgc.server.gcInterval=9223372036854775806
   Class-Path:
   /Users/username/vmware-gemfire-9.10.17/lib/geode-core-1.15.0.jar:/gemfire/tool
   s/Modules/gemfire-prometheus-metrics/gson-2.8.6.jar:/gemfire/tools/Modules/gemf
   ire-prometheus-metrics/gemfire-prometheus-metrics-0.1.0-build.86.jar:/gemfire/t
   ools/Modules/gemfire-prometheus-metrics/micrometer-registry-prometheus-1.6.3.ja
   r:/gemfire/tools/Modules/gemfire-prometheus-metrics/simpleclient_common-0.9.0.j
   ar:/gemfire/tools/Modules/gemfire-prometheus-metrics/simpleclient-0.9.0.jar:/Us
   ers/username/vmware-gemfire-9.10.17/lib/geode-server-all-1.15.0.jar

   Successfully connected to: JMX Manager [host=192.168.50.245, port=1099]

   Cluster configuration service is up and running.
   ```

3. Verify that the metrics module is properly configured by visiting the following URL:

   ```
   http://HOSTNAME:PORT/metrics
   ```

   Where:

   - **HOSTNAME** is the IP address of the member that you recorded in a previous step.
   - **PORT** is the metrics port that you specified with the command-line parameter `gemfire.prometheus.metrics.port` in a previous step.

   Using the example information above, this would be: `http://192.168.50.245:7001/metrics`.

   The output should resemble the following:

```
# HELP gemfire_replyWaitTime
# TYPE gemfire_replyWaitTime gauge
gemfire_replyWaitTime{category="DistributionStats",instance="distributionStat
s",member="192.168.
129.137(locator1:76435:locator)<ec><v0>:41000",} 0.0
# HELP gemfire_loadsCompleted
# TYPE gemfire_loadsCompleted gauge
gemfire_loadsCompleted{category="CachePerfStats",instance="RegionStats-manageme
ntRegionStats",member="192.168.129.137(locator1:76435:locator)<ec><v0>:41000",}
0.0
gemfire_loadsCompleted{category="CachePerfStats",instance="cachePerfStats",memb
er="192.168.129.137(locator1:76435:locator)<ec><v0>:41000",} 0.0
...
```

Example

The following example enables Wavefront-viewable metrics on a single GemFire locator, then shows how to scrape metrics using Telegraf and view the results using Wavefront. The example uses the MacOS-specific `brew` command. You may need to adapt it for use on other platforms.

The example configuration contains two main parts:

- GemFire setup
- Telegraf and Wavefront Proxy setup (metrics collection agent and forwarder)

GemFire Setup

The example requires that some GemFire metrics be enabled from a `gemfire.properties` file.

1. Create a file named gemfire.properties with the following content:

```
statistic-sampling-enabled=true
statistic-archive-file=stats.gfs
enable-time-statistics=true
```

2. To simplify command lines, set two environment variables to the paths for use in the code snippets below. Paths shown here are placeholders. Substitute the paths that match your system.

```
METRICS_PATH="~/gemfire/tools/Modules/gemfire-prometheus-metrics/*"
GEMFIRE_PROPERTIES_FILE_PATH="./gemfire.properties"
```

With these configuration parameters in place, you can start GemFire using `gfsh` or Launcher, as shown below.

Example GemFire Startup Using gfsh

```
gfsh -e "start locator --name=locator1 --port=10334 \
  --classpath=$METRICS_PATH \
  --properties-file=$GEMFIRE_PROPERTIES_FILE_PATH \
  --J=-Dgemfire.prometheus.metrics.port=7001"
```

Example GemFire Startup Using Launcher

Launcher startup requires an explicit path to the `gemfire-dependencies.jar` file. Substitute the path that matches your system.

```
java -classpath \
"$METRICS_PATH:/gemfire/lib/geode-dependencies.jar" \
-DgemfirePropertyFile="$GEMFIRE_PROPERTIES_FILE_PATH" \
-Dgemfire.prometheus.metrics.port=7001 \
org.apache.geode.distributed.LocatorLauncher start locator --port=10334 &
```

Telegraf and Wavefront Proxy Setup

This example uses Telegraf as the agent to pull Wavefront-viewable Prometheus-style metrics from the GemFire cluster. It sends them to a local Wavefront proxy, which forwards them to the Wavefront service.

Install Telegraf and the Wavefront Proxy, as described on the Wavefront website:

- Linux (tip: on Linux, the wave cli can install both)
- Mac
- Windows

Acquire an API token that will allow Tanzu Observability to authenticate communication from the Wavefront Proxy. Follow the directions at Generating an API Token.

Configuring Telegraf

Once installed, add a config file with the specified tags and the urls to indicate to Telegraf to scrape metrics from GemFire's prometheus-style endpoints. Here, the configuration file is named "tanzu-gemfire.conf":

```
# Telegraf config to scrape GemFire metrics
[agent]
  interval = "10s"
[[inputs.prometheus]]
  urls = ["http://localhost:7001/metrics","http://localhost:8001/metrics"]
  # These tags are used in the Wavefront-GemFire integration. Set them to uniquely ide
ntify your GemFire cluster.
  [inputs.prometheus.tags]
    "label.gemfire-environment" = "milky-way"
    "label.gemfire-cluster" = "my-cluster"
[[outputs.wavefront]]
    host = "localhost"
    port = 2878
    metric_separator = "."
    source_override = ["hostname", "agent_host", "node_host"]
    convert_paths = true
    use_regex = false
```

Set an environment variable so telegraf can find the configuration file:

```
export TELEGRAF_CONFIG_PATH=<path-to-config-file>/tanzu-gemfire.conf
```

Restart the wavefront proxy:

```
brew services restart wfproxy
```

Start telegraf:

```
brew services restart telegraf
```

In a browser, navigate to your GemFire dashboard on Wavefront. You should see live metrics. To find your Wavefront dashboard, see VMware Tanzu Observability.

As an alternative, you can view the logfile in a shell window:

```
tail -f /usr/local/var/log/wavefront/wavefront.log
```

Output should resemble:

```
2021-06-02 11:59:20,210 INFO  [proxy:checkin] Checking in: https://vmware.wavefront.co
m/api
2021-06-02 11:59:20,210 INFO  [proxy:checkin] Checking in: https://vmware.wavefront.co
m/api
2021-06-02 11:59:29,915 INFO  [AbstractReportableEntityHandler:printStats] [2878] Poin
ts received rate: 302 pps (1 min), 296 pps (5 min), 1588 pps (current).
2021-06-02 11:59:29,915 INFO  [AbstractReportableEntityHandler:printStats] [2878] Poin
ts received rate: 302 pps (1 min), 296 pps (5 min), 1588 pps (current).
2021-06-02 11:59:29,915 INFO  [AbstractReportableEntityHandler:printStats] [2878] Poin
ts delivered rate: 295 pps (1 min), 294 pps (5 min)
2021-06-02 11:59:29,915 INFO  [AbstractReportableEntityHandler:printStats] [2878] Poin
ts delivered rate: 295 pps (1 min), 294 pps (5 min)
```

Verification and Troubleshooting suggestions

If everything is working properly, your cluster should be listed within the Tanzu GemFire integration in Wavefront.

If everything is not working properly, try these suggestions:

- GemFire
  - Verify the metrics endpoint is hosting metrics by curling one of the metrics endpoints or viewing it in your browser, e.g. `curl localhost:7001/metrics`.
  - View the member's log and verify the metrics module is loaded.
- Telegraf
  - Try viewing its logs or starting it in console mode (for example, `.\telegraf --console install` in Windows) to catch any suppressed errors.
  - Ensure the scraped urls contain the correct metrics ports.
- Wavefront Proxy
  - Verify that the Wavefront Proxy logs are actively receiving datapoints.
  - Check that Wavefront Proxy has the correct subdomain (`<your-subdomain>.wavefront.com`) and a valid API key.

Default Dashboard Metrics

These tables list the metrics available in the default Wavefront dashboard.

Tanzu GemFire Cluster Metrics

Metric	Description
Uptime	Maximum uptime of all members in this cluster
Server Count	Count of members that are servers and serve up data in this cluster
Locator Count	Count of locators in this cluster. The locator is a Tanzu GemFire process that tells new, connecting members where running members are located and provides load balancing for server use.
Region Count	Count of regions in this Tanzu GemFire cluster. Regions hold entries (key/value pairs) in buckets.
CPU Count	The current count of CPUs recognized by Tanzu GemFire
Cluster Entries	This metric represents the last value registered for the total entry count across all regions and data in the specified Tanzu GemFire cluster. For partitioned data, we take a raw sum of the region entries. For replicate data, we only count the primary copy and only use the result from one server.
Server Old Gen Utilization	For each server in this cluster, this chart shows the amount of memory, in bytes, being used by each server and the overall % of that member's available memory usage on the right axis.
Disk Utilization	Disk utilization by member in the cluster. The percentage remaining is computed by using the statistics gathered by Tanzu GemFire from the underlying system. Some operating systems (like macOS) do not emit these metrics and will cause this graph to be empty.
CPU Utilization % by Member	The current utilization percentage of this cluster's CPUs
Current Client Connects by Member	This chart displays a line for each server in the cluster and represents the number of external clients connected to that member.
Sampler Delay Duration	Delay duration measures the actual time (in ms) the Tanzu GemFire statistics sampler slept. It calculates the difference between when it went to sleep and when it woke up to sample. Sample time shows how long it took to collect the sample after it woke up.
Cluster Communication Delays	Tanzu GemFire uses 'replyWaitsInProgress' as a means to measure intra-cluster communication and determine a stalled or failing member.
IO Waits	The time spent waiting to read or write
Abandoned Reads/Second	Tanzu GemFire monitors the AbandonedReadRequests from the cache server.
CPU Steal Time	Steal time is the percentage of time a virtual CPU waits for a real CPU while the hypervisor is servicing another virtual processor. Your virtual machine (VM) shares resources with other instances on a single host in a virtualized environment. One of the resources it shares is CPU Cycles.

Tanzu GemFire Features Metrics

Metric	Description
Gets Throughput	This chart displays the throughput of all ‘Get’ operations performed across the entire cache. RIGHT Axis (RED): The ‘Get’ operation requests by the remote Client applications. LEFT Axis (BLUE): The ‘Get’ operation requests by the remote Client applications and distributed get operations.
Average Get Latency	Average Get Latency presents the total time taken by all ‘Get’ operations performed across the cluster divided by the number of all ‘Get’ operations performed across the cluster to get an average time taken per ‘Get’ operation in the system.
Cache Hits Per Second	The rate of Get operations across the entire cluster that resulted in a matched key per second.
Cache Misses Per Second	This chart represents the rate of Get operations that resulted in a miss, where the key that was requested was not in the cache per second.
Puts Throughput	This chart displays the throughput of all ‘Put’ operations performed across the entire cache. RIGHT Axis (RED): The ‘Put’ operation requests by the remote Client applications. LEFT Axis (BLUE): The ‘Put’ operation requests by the remote Client applications and distributed ‘Put’ operations
Average Put Latency	Average Put Latency presents the total time taken by all ‘Put’ operations performed across the cluster divided by the number of all ‘Put’ operations performed across the cluster to get an average time taken per ‘Put’ operation in the system.
Cache Hit Ratio	The Cache hit ratio is currently based on system ‘Get’ operations and any ‘Get’ operation across the cluster. It represents the percentage of those ‘Get’ operations that return a value based on the specified key.
Function Execution Details	RED (RIGHT AXIS): Function Execution Queue size. BLUE (LEFT AXIS): A calculated rate based on functionExecutionCalls minus functionExecutionsCompleted, which provides the output of FAILED function execution calls per second.
Client Put Request By Server	For each server member, this chart represents the rate of requests of ‘Put’ operations received per second on a specific member.
Persistent Region Overcapacity	For persistent regions that hold data in memory and on disk, we provide this chart to show whether the capacity of system is sufficient to accommodate the stored data.
Message Queue Size	Size of the Tanzu GemFire message queue in number of operations (such as ‘Get’, ‘Put’, and ‘Destroy’) that it contains
Client Query Rate	The number of queries received per second from GemFire clients
Average Query Time	The average query time is computed by taking the total time of all query requests (processQueryTime) and dividing it by the number of requests (queryRequests), to get an average time spent per query.
Region Details	This table shows the type, name, and entry count for each region in the cache. For convenience, we have summarized the Partitioned regions with the Partition Total rows.

WAN Gateway Metrics

Metric	Description
WAN Receiver Throughput	This chart shows all of the Gateway Receivers for this cluster and the rate of bytes/second that are sent (red) or received (blue).
WAN Sender Queue	This graph displays the rate (messages/second) at which a Gateway Sender is able to send events to a WAN connected cluster.

VMware GemFire Pulse

VMware GemFire Pulse is a Web Application that provides a graphical dashboard for monitoring vital, real-time health and performance of VMware GemFire clusters, members, and regions.

Use Pulse to examine total memory, CPU, and disk space used by members, uptime statistics, client connections, WAN connections, and critical notifications. Pulse communicates with a VMware GemFire JMX manager to provide a complete view of your VMware GemFire deployment. You can drill down from a high-level cluster view to examine individual members and even regions within a member, to filter the type of information and level of detail.

By default, VMware GemFire Pulse runs in an embedded container within a VMware GemFire JMX manager node. You can optionally deploy Pulse to a Web application server of your choice, so that the tool runs independently of your VMware GemFire clusters. Hosting Pulse on an application server also enables you to use SSL for accessing the application.

- **Pulse System Requirements**

  Verify that your system meets the installation and runtime requirements for GemFire Pulse.

- **Running Pulse in Embedded Mode (Quick Start)**

  Use Pulse in embedded mode to monitor a VMware GemFire deployment directly from a VMware GemFire JMX Manager. By default, the embedded Pulse application connects to the local JMX Manager that hosts the Pulse application. Optionally, configure Pulse to connect to a VMware GemFire system of your choice.

- **Hosting Pulse on a Web Application Server**

  Host Pulse on a dedicated Web application server to make the Pulse application available at a consistent address, or to use SSL for accessing the Pulse application. When you host Pulse in this way, you also configure Pulse to connect to a specific locator or JMX Manager node for monitoring.

- **Configuring Pulse Authentication**

  Pulse requires all users to authenticate themselves before they can use the Pulse Web application. If you have configured JMX authentication on the VMware GemFire JMX Manager node, the Pulse Web application itself may also need to authenticate itself to the VMware GemFire JMX Manager node on startup.

- **Using Pulse Views**

  Pulse provides a variety of different views to help you monitor VMware GemFire clusters, members, and regions.

Pulse System Requirements

Verify that your system meets the installation and runtime requirements for Pulse.

The Pulse Web application has been tested for compatibility with the following Web browsers:

- Internet Explorer 9.0.8112.16421
- Safari 5.1.7 for Windows
- Google Chrome 22.0.1229.79 m
- Mozilla Firefox 16.0.1

Pulse has been tested for standalone deployment on Tomcat and Jetty. Pulse may work with other operating systems and browsers upon which it has not been tested.

Running Pulse in Embedded Mode (Quick Start)

Use Pulse in embedded mode to monitor a VMware GemFire deployment directly from a VMware GemFire JMX Manager. By default, the embedded Pulse application connects to the local JMX Manager that hosts the Pulse application. Optionally, configure Pulse to connect to a VMware GemFire system of your choice.

To run Pulse in embedded mode:

1. Configure a VMware GemFire member to run as a JMX Manager node, specifying the HTTP port on which you will access the Pulse Web application (port 7070 by default). For example, the following command starts a VMware GemFire locator as a JMX Manager node, using the default HTTP port 7070 for the Pulse application:

   ```
   gfsh
   gfsh> start locator --name=loc1
   ```

   Note: VMware GemFire locators become JMX Manager nodes by default. To start a non-locator member as a JMX Manager node, include the `--J=-Dgemfire.jmx-manager=true` option. To specify a non-default port number for the HTTP service that hosts the Pulse application, include the `--J=-Dgemfire.http-service-port=port_number` option when starting the JMX Manager node.

   When the JMX Manager node boots, it starts an embedded Jetty instance and deploys the Pulse Web application at the specified or default HTTP port or 7070 by default.

   `gfsh` automatically connects to the manager when you start it in this way. If you already started a manager process earlier, use the `connect` command in `gfsh` to connect to that process.

2. Access the embedded Pulse application from a Web browser. If you are connected to the VMware GemFire cluster using gfsh, use the `start pulse` command to load the correct URL in your browser:

   ```
   gfsh> start pulse
   ```

   Or, enter the URL http://*address*:*http-service-port*/pulse directly in your Web browser, substituting the address and HTTP port of the manager. For example, you access Pulse on the local locator machine from Step 1 at the URL http://localhost:7070/pulse.

3. If you have configured authentication for the Pulse application, enter the username and password of a valid Pulse account in the login screen. Otherwise, enter the default “admin” in both fields. Click **Sign In** to continue.

   See Configuring Pulse Authentication.

4. After you log in, Pulse displays the main cluster view for the local cluster. See Using Pulse Views.

Note: When running in embedded mode, the Pulse application connects only to the JMX Manager running in the locator or member that hosts Pulse. This enables you to monitor all members of that cluster. You can also view (but not monitor) connected WAN clusters, and can view gateway senders and receivers that are configured in the local cluster.

Hosting Pulse on a Web Application Server

Host Pulse on a dedicated Web application server to make the Pulse application available at a consistent address, or to use SSL for accessing the Pulse application. When you host Pulse in this way, you also configure Pulse to connect to a specific locator or JMX Manager node for monitoring.

To host Pulse on a Web application server:

1. Set the `http-service-port` property to zero (`-Dgemfire.http-service-port=0`) when you start your VMware GemFire JMX Manager nodes. Setting this property to zero disables the embedded Web server for hosting the Pulse application.

2. Deploy the Pulse Web application to your application server. VMware GemFire installs the `geode-pulse-n.n.n.war` file (where `n.n.n` is a version number) in the `tools/Pulse` subdirectory of your VMware GemFire installation directory. Depending on your application server, you may need to copy the `pulse.war` file to a deployment directory or use a configuration tool to deploy the file.

3. Stop the Web application server and locate the Pulse configuration in the `WEB-INF/classes` subdirectory.

4. Edit `pulse.properties`, defining or redefining any of the following configuration properties as needed for your application:

Property	Description
`pulse.useLocator`	Specify "true" to configure Pulse to connect to a VMware GemFire Locator member, or "false" to connect directly to a JMX Manager. When Pulse connects to a VMware GemFire locator, the locator provides the address and port of an available JMX Manager to use for monitoring the cluster. In most production deployments, you should connect Pulse to a locator instance; this allows Pulse to provide monitoring services using any available JMX Manager. If you specify "false," Pulse connects directly to a specific JMX Manager. If this manager is not available, the Pulse connection fails, even if another JMX Manager is available in the cluster.
`pulse.host`	Specify the DNS name or IP address of the VMware GemFire locator or JMX Manager machine to which Pulse should connect. You specify either a locator or JMX Manager address depending on how you configured the `pulse.useLocator` property.
`pulse.port`	Specify the port number of the VMware GemFire locator or the HTTP port number of the JMX Manager to which Pulse should connect. You specify either a locator or JMX Manager port depending on how you configured the `pulse.useLocator` property. If you configured `pulse.useLocator=false`, then `pulse.port` must correspond to the `http-service-port` setting of the JMX Manager.

For example, with the default `pulse.properties` configuration, Pulse connects to the locator at port 10334 and accesses any available JMX Manager:

```
pulse.useLocator=true
pulse.host=localhost
pulse.port=10334
```

With this modified configuration, Pulse accesses only the JMX Manager instance at port 8080:

```
pulse.useLocator=false
pulse.host=jmxsrv.mycluster.com
pulse.port=8080
```

5. If a JMX manager or locator is configured to use SSL, you can configure Pulse to connect to these processes. Edit `pulsesecurity.properties` to un-comment the standard Java SSL properties:

```
javax.net.ssl.keyStore={KeyStorePath}
javax.net.ssl.keyStorePassword={KeyStorePassword}
javax.net.ssl.trustStore={TrustStorePath}
javax.net.ssl.trustStorePassword={TrustStorePassword}
```

Substitute the appropriate paths and passwords for the bracketed placeholders.

6. To change Pulse logging behavior, edit `log4j2.xml` in the same directory. The default configuration sets the logging level to `INFO` and specifies a message pattern that includes the date, name of the reporting logger, error level, name of the reporting thread, and the error message. Refer to the Apache Log4j documentation for details on how to specify `log4j2.xml` content and syntax.
7. Restart the Web application server.
8. Access the Pulse application using the address, port, and application URL that you configured in your Web application server. For example, with Tomcat the default URL is `http://*address*:8080/pulse`. Your application server provides options for configuring the address, port, and application name; substitute the correct items to access the deployed Pulse application.

   Pulse connects to the locator or JMX Manager that you configured in the `pulse.properties` file, authenticating using the credentials that you configured in the file.
9. If you have configured authentication for the Pulse application, enter the username and password of a valid Pulse account in the login screen. Otherwise, enter the default “admin” in both fields. Click **Sign In** to continue.

   See Configuring Pulse Authentication.
10. After you log in, Pulse displays the main cluster view for the distributed system to which it has connected. See Using Pulse Views.

Configuring Pulse Authentication

Pulse requires all users to authenticate themselves before they can use the Pulse Web application.

If you run Pulse in embedded mode, the Pulse application runs on the JMX Manager node and no JMX authentication is required. You do not need to specify valid JMX credentials to start an embedded Pulse application.

If you host Pulse on a web application server (non-embedded mode) and you configure JMX authentication on the VMware GemFire manager node, then the Pulse Web application must authenticate itself with the manager node when it starts. Specify the credentials of a valid JMX user account in the `pulse.properties` file, as described in Hosting Pulse on a Web Application Server.

Note: The credentials that you specify must have both read and write privileges in the JMX Manager node. See Configuring a JMX Manager.

Configuring Pulse to use HTTPS

You can configure Pulse to use HTTPS in either embedded or non-embedded mode.

Embedded Mode

In embedded mode, VMware GemFire uses an embedded Jetty server to host the Pulse Web application. To make the embedded server use HTTPS, you must enable the `http` SSL component in `gemfire.properties` or `gfsecurity.properties`. See SSL for details on configuring these parameters.

These SSL parameters apply to all HTTP services hosted on the JMX Manager, which includes the following:

- Developer REST API service
- Management REST API service (for remote cluster management)
- Pulse monitoring tool

When the `http` SSL component is enabled, all HTTP services become SSL-enabled and you must configure your client applications accordingly. For SSL-enabled Pulse, you will need to configure your browsers with proper certificates.

Non-Embedded (Standalone Web Server) Mode

In non-embedded mode where you are running Pulse on a standalone web application server, such as Tomcat, you must use the web server's SSL configuration to make the HTTP requests secure.

Configuring Pulse to use Security Manager

You can configure Pulse to use the VMware GemFire Security Manager in either embedded or non-embedded mode.

Embedded Mode

To use Security Manager with Pulse running in embedded mode, you do not need to specify additional credentials. Pulse will automatically be configured by VMware GemFire to send the credentials entered in the Pulse login page to the Security Manager for authentication and authorization.

Non-Embedded (Standalone Web Server) Mode

When running Pulse on a standalone web application server, such as Tomcat, you need to configure the Pulse Web app to use a VMware GemFire-specific security profile. Activate the VMware GemFire profile at startup with the system property:

```
-Dspring.profiles.active=pulse.authentication.gemfire
```

Configuring Pulse to use a Custom Security Profile

You can use a custom security profile only if you are NOT using a Security Manager in your cluster. A custom security profile only controls who can access the Pulse Web page. You will need to create an XML file named `pulse-authentication-custom.xml` that defines the credentials and roles of the users of Pulse.

Example pulse-authentication-custom.xml for Spring simple in-memory User Details Service

```
<beans:beans >
  <authentication-manager>
    <authentication-provider>
      <user-service id="userDetailsService">
        <user name="john" password="johnspassword" authorities="ROLE_CLUSTER:READ" />
        <user name="bob" password="bobspassword" authorities="ROLE_CLUSTER:READ" />
      </user-service>
    </authentication-provider>
  </authentication-manager>
</beans:beans>
```

Embedded Mode

Put `pulse-authentication-custom.xml` in the locator's working directory. When you start VMware GemFire members, specify the custom authentication profile using the `-Dspring.profiles.active=pulse.authentication.custom` system property. For example:

```
gfsh> start locator --name=locator --J=-Dspring.profiles.active=pulse.authentication.c
ustom
```

Start Pulse and log in using credentials that are authorized in the custom configuration.

Non-Embedded (Standalone Web Server) Mode

Put `pulse-authentication-custom.xml` in the Pulse Web app's classpath, and start the app with the system property:

```
-Dspring.profiles.active=pulse.authentication.custom
```

Start Pulse and log in using credentials that are authorized in the custom configuration.

Configuring Pulse to use an OAuth Authentication Provider

You can configure Pulse to use any Authentication Provider that supports OpenID and custom scopes. You will need to:

- Register with the authentication provider, obtaining a valid client ID and client secret for Pulse.
- Configure the scopes with `CLUSTER:READ`, `CLUSTER:WRITE`, `DATA:READ` or `DATA:WRITE`.

Pulse Properties for OAuth Configuration

After you set up the authentication provider properly, create a properties file named `pulse.properties` (or edit it, if it already exists) to define the following required properties:

- **pulse.oauth.providerId**

  An ID for Pulse to use internally to identify your OAuth provider.
- **pulse.oauth.providerName**

  A name for Pulse to display on the login page to link to your OAuth provider.
- **pulse.oauth.clientId**

  The ID with which Pulse is registered with your OAuth provider.
- **pulse.oauth.clientSecret**

  The secret with which Pulse is registered with your OAuth provider.
- **pulse.oauth.authorizationUri**

  The URI for your OAuth provider's authorization endpoint.

- **pulse.oauth.tokenUri**

  The URI for your OAuth provider's token endpoint.

- **pulse.oauth.userInfoUri**

  The URI for your OAuth provider's user info endpoint.

- **pulse.oauth.jwkSetUri**

  The URI for your OAuth provider's JSON Web Key (JWK) Set endpoint.

- **pulse.oauth.endSessionEndpoint**

  The URI for your OAuth provider's endpoint to request that the end user be logged out. See the `end_session_endpoint` parameter of the OpenID Provider Discovery Metadata standard proposal.

- **pulse.oauth.userNameAttributeName**

  The attribute name used to access the user's name from your OAuth provider's user info response.

Example pulse.properties for OAuth Configuration

```
pulse.oauth.providerId=uaa
pulse.oauth.providerName=UAA
pulse.oauth.clientId=pulse
pulse.oauth.clientSecret=example-pulse-client-secret
pulse.oauth.authorizationUri=http://example.com/uaa/oauth/authorize
pulse.oauth.tokenUri=http://example.com/uaa/oauth/token
pulse.oauth.userInfoUri=http://example.com/uaa/userinfo
pulse.oauth.jwkSetUri=http://example.com/uaa/token_keys
pulse.oauth.endSessionEndpoint=http://example.com/uaa/profile
pulse.oauth.userNameAttributeName=user_name
```

Embedded Mode

1. Create the above `pulse.properties` file and put it in the member's working directory.
2. Start up the member with a VMware GemFire property (either specify it in `gemfire.properties` file or pass it in at startup) `security-auth-token-enabled-components=pulse` or `security-auth-token-enabled-components=all`.

```
gfsh> start locator --name=locator --J=-Dgemfire.security-auth-token-enabled-component
s=pulse
```

Start pulse and you will see login page being redirected to the configured authentication provider.

Non-Embedded (Standalone Web Server) Mode

1. Edit the existing `pulse.properties` file and add the above content, make sure the file is in the web app's classpath
2. Start the app with the system property:

```
-Dspring.profiles.active=pulse.authentication.oauth
```

Start Pulse and you will see login page being redirected to the configured authentication provider.

Using Pulse Views

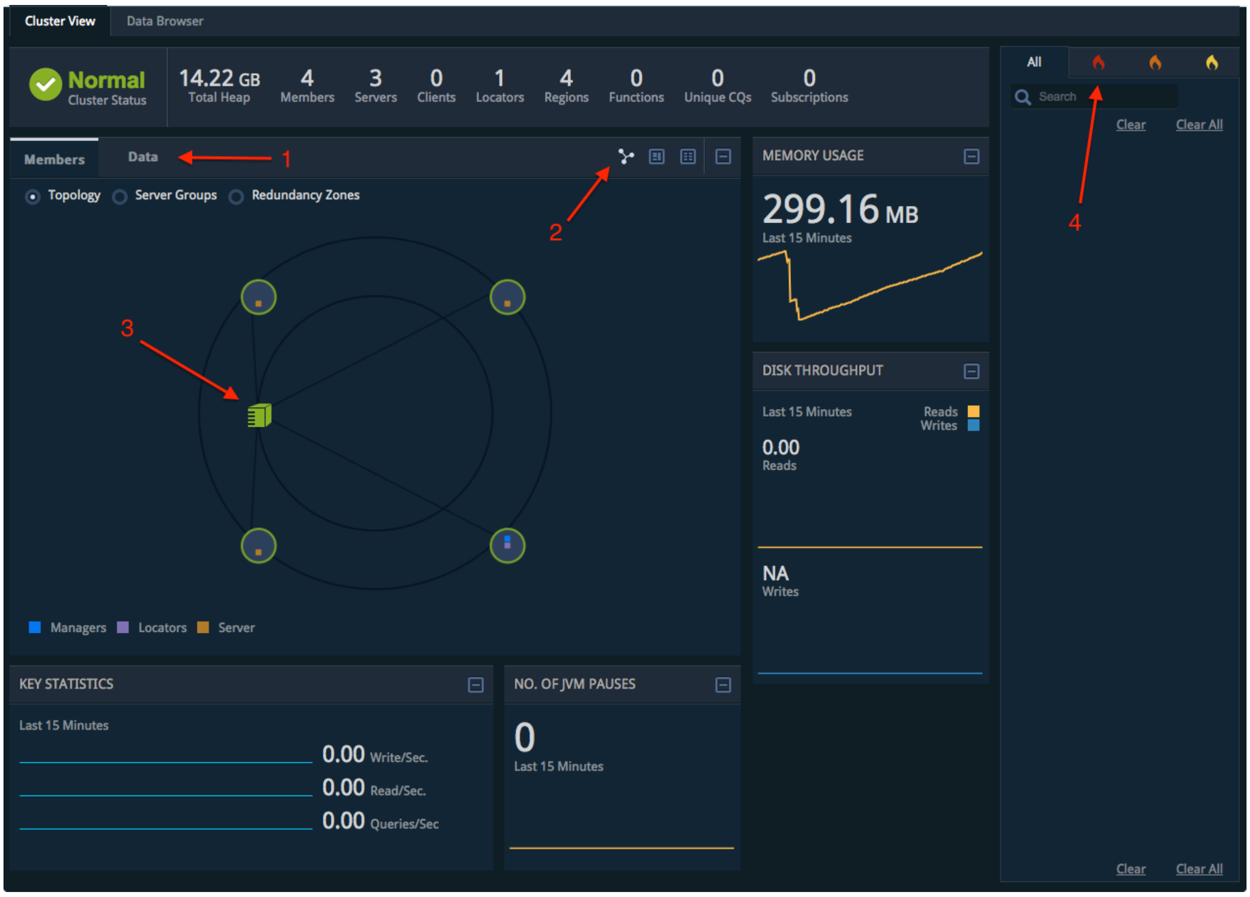
Pulse provides a variety of different views to help you monitor VMware GemFire clusters, members, and regions.

The following sections provide an overview of the main Pulse views:

- Cluster View
- Member View
- Region View
- Data Browser
- Alerts Widget

Cluster View

The cluster view is a high-level overview of the cluster. It is displayed immediately after you log into Pulse. Information displays around the perimeter of the cluster view show statistics such as memory usage, JVM pauses, and throughput. You can use the cluster view to drill down into details for individual members and regions in the cluster.

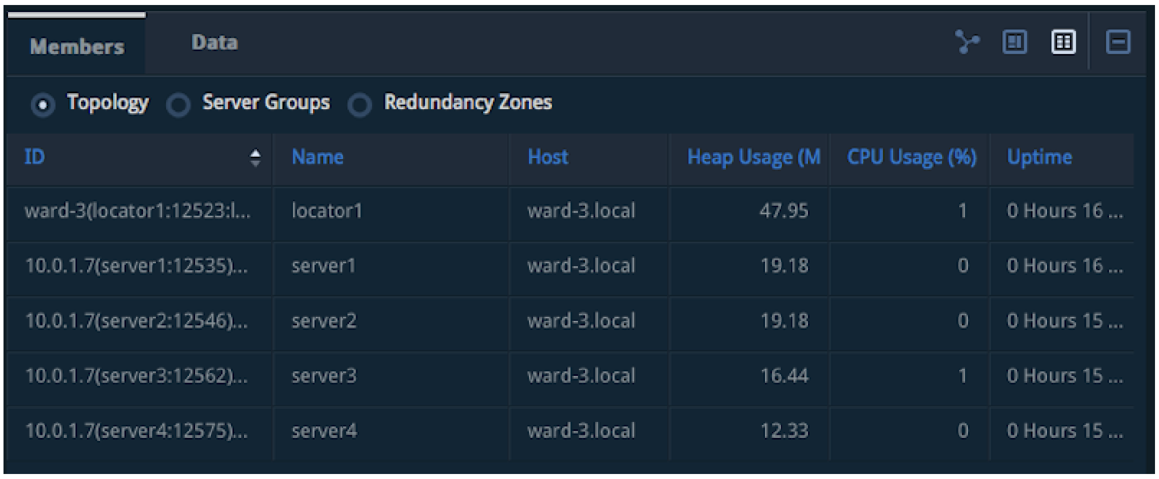


Use these basic controls while in Cluster view:

1. Click Members or Data to display information about VMware GemFire members or data regions in the cluster.

2. Click the display icons to display the VMware GemFire members using icon view, block view, or table view. Note that icon view is available only when displaying Members.

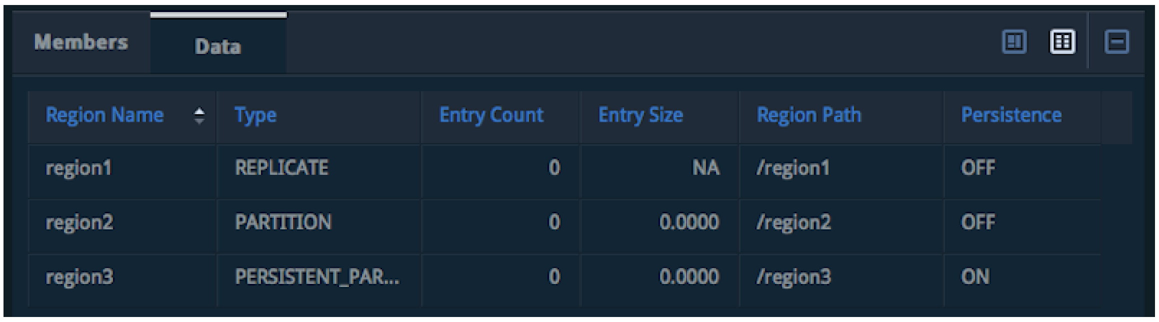
   For example, the following shows VMware GemFire Members displayed in table view:



ID	Name	Host	Heap Usage (M	CPU Usage (%)	Uptime
ward-3(locator1:12523:l...	locator1	ward-3.local	47.95	1	0 Hours 16 ...
10.0.1.7(server1:12535)...	server1	ward-3.local	19.18	0	0 Hours 16 ...
10.0.1.7(server2:12546)...	server2	ward-3.local	19.18	0	0 Hours 15 ...
10.0.1.7(server3:12562)...	server3	ward-3.local	16.44	1	0 Hours 15 ...
10.0.1.7(server4:12575)...	server4	ward-3.local	12.33	0	0 Hours 15 ...

   - While in block view or table view, click the name of a VMware GemFire member to display additional information in the Member View.
   - Click Topology, Server Groups, or Redundancy Zones to filter the view based on all members in the topology, configured server groups, or configured redundancy zones.

   The following shows VMware GemFire Regions displayed in table view:



Region Name	Type	Entry Count	Entry Size	Region Path	Persistence
region1	REPLICATE	0	NA	/region1	OFF
region2	PARTITION	0	0.0000	/region2	OFF
region3	PERSISTENT_PAR...	0	0.0000	/region3	ON

   - While in block view or table view, click the name of a VMware GemFire region to display additional information in the Region View.

3. While in icon view, click a host machine icon to display the VMware GemFire members on that machine.
4. In the Alerts pane, click the severity tabs to filter the message display by the level of severity.

Cluster View Screen Components

The following table describes the data pieces displayed on the Cluster View screen.

Screen Component	Description
Cluster Status	Overall status of the cluster being monitored. Possible statuses include Normal, Warning, or Severe.
Total Heap	Total amount of memory (in GB) allocated to the Java heap across all members.

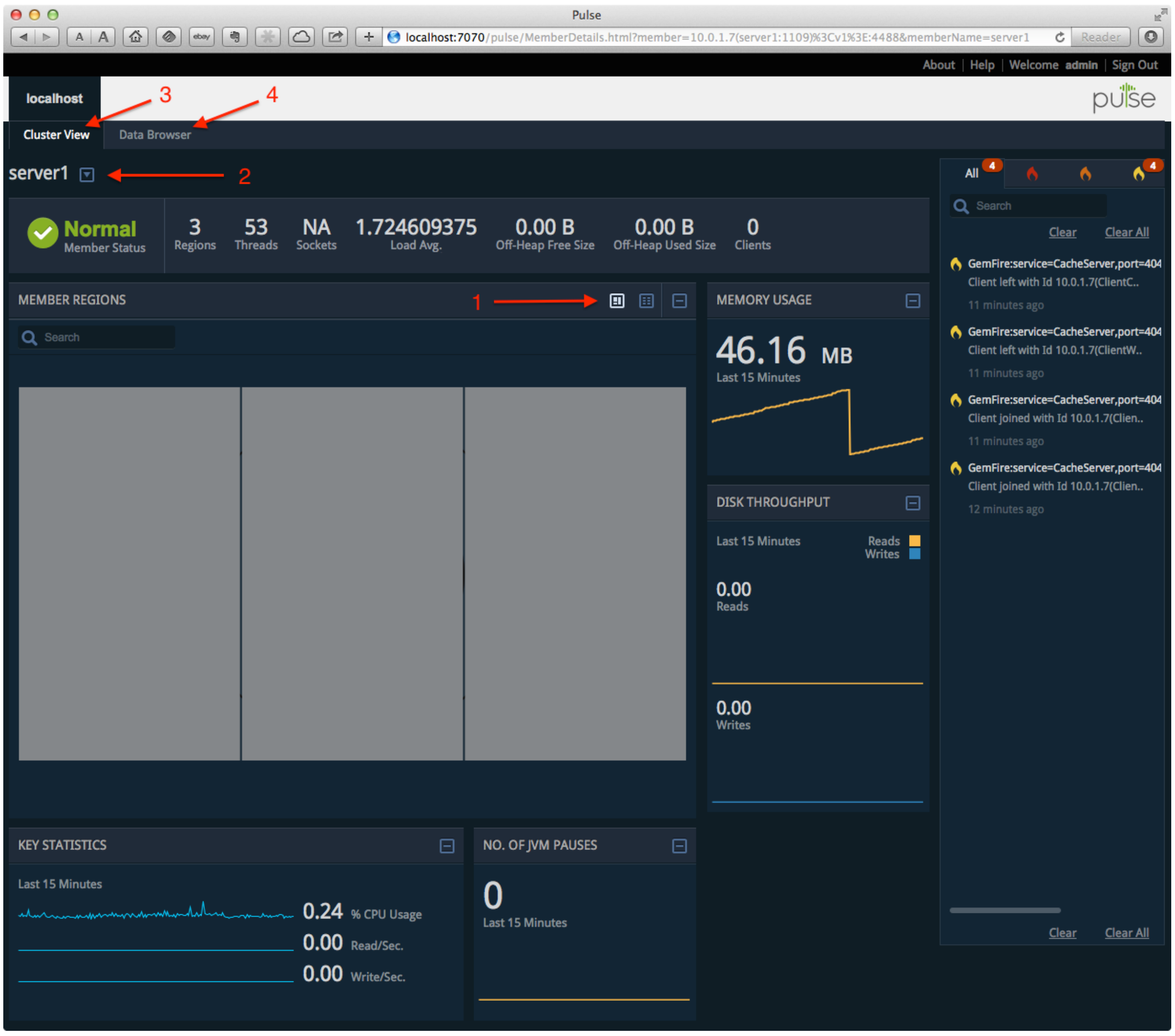
Screen Component	Description
Members	Total number of members in the cluster.
Servers	Total number of servers in the cluster.
Clients	Total number of clients in the cluster.
Locators	Total number of locators in the cluster.
Regions	Total number of regions in the cluster.
Functions	Total number of functions registered in the cluster.
Unique CQs	Total number of unique CQs. Corresponds to the UNIQUE_CQ_QUERY statistic.
Subscriptions	Total number of client event subscriptions.
Cluster Members	Graphical, block, or table view of the members in the cluster.
Topology	Organizes cluster members by DistributedMember Id.
Server Groups	Organizes cluster members by server group membership. If no server groups are configured, all members appear under the "Default" server group.
Redundancy Zones	Organizes cluster members by redundancy zones. If no redundancy zones are configured, all members appear under the "Default" zone.
Host Machine	When you mouse over a machine icon in Topology View, a pop-up appears with the following machine statistics: • *CPU Usage*. Percentage of CPU being used by VMware GemFire processes on the machine. • *Memory Usage*. Amount of memory (in MB) being used by VMware GemFire processes. • *Load Avg*. Average number of threads on the host machine that are in the run queue or are waiting for disk I/O over the last minutes. Corresponds to the Linux System statistic loadAverage1. If the load average is not available, a negative value is shown. • *Sockets*. Number of sockets currently open on the machine.

Screen Component	Description
Member	When you mouse over a member icon in Graphical View, a pop-up appears with the following member statistics: • *CPU Usage*. Percentage of CPU being used by the VMware GemFire member process. • *Threads*. Number of threads running on the member. • *JVM Pauses*. Number of times the JVM used by the member process has paused due to garbage collection or excessive CPU usage. • *Regions*. Number of regions hosted on the member process. • *Clients*. Number of client currently connected to the member process. • *Gateway Sender*. Number of gateway senders configured on the member. • *Port*. Server port of the cache server member where clients can connect and perform cache operations. • *GemFire Version*. The version of the VMware GemFire member.
Member	In List View, the following data fields are displayed for each member: • *ID*. DistributedMember Id of the member. • *Name*. Name of the member. • *Host*. Hostname or IP address where the member is running. • *Heap Usage*. Amount of JVM heap memory being used by the member process. • *CPU Usage*. Percentage of CPU being used by the VMware GemFire member process. • *Uptime*. How long the member has been up and running. • *Clients*. Number of clients currently connected to the member. It will have a value only if the member acts as a CacheServer.
Key Statistics	Displays a few key performance measurements of the cluster (over the last 15 minutes).
Write/Sec	Number of write operations per second that have occurred across the cluster. Each put/putAll operation counts as a write; for example, a putAll of 50 entries is counted as one write.
Read/Sec	Number of read operations per second that have occurred across the cluster.
Queries/Sec	Number of queries per second that have been executed across the cluster.

Screen Component	Description
No. of JVM Pauses	Number of times the JVM has paused during the last five minutes to perform garbage collection.
WAN Information	If you have configured gateway senders or receivers for a multi-site (WAN) deployment, this box displays whether the remote cluster is reachable (working connectivity represented by a green triangle).
Disk Throughput	Total disk throughput for all disks in cluster.
Alerts View	Displays alerts for the cluster.

Member View

When you select an individual VMware GemFire member in Cluster View, Pulse displays the regions available on that member, as well as member-specific information such as the configured listen ports.



GemFire:service=CacheServer,port=404
Client left with Id 10.0.1.7(ClientC..
11 minutes ago

GemFire:service=CacheServer,port=404
Client left with Id 10.0.1.7(ClientW..
11 minutes ago

GemFire:service=CacheServer,port=404
Client joined with Id 10.0.1.7(Clien..
11 minutes ago

GemFire:service=CacheServer,port=404
Client joined with Id 10.0.1.7(Clien..
12 minutes ago

Use these basic controls while in Member View:

1. Click the display icons to display regions using block view or table view.

2. Use the drop down menu to select a specific member or search for specific members by name.
3. Click **Cluster View** to return to Cluster View. See Cluster View.
4. Click **Data Browser** to query region data. See Data Browser.

Member View Screen Components

The following table describes the data elements displayed on the Member View screen.

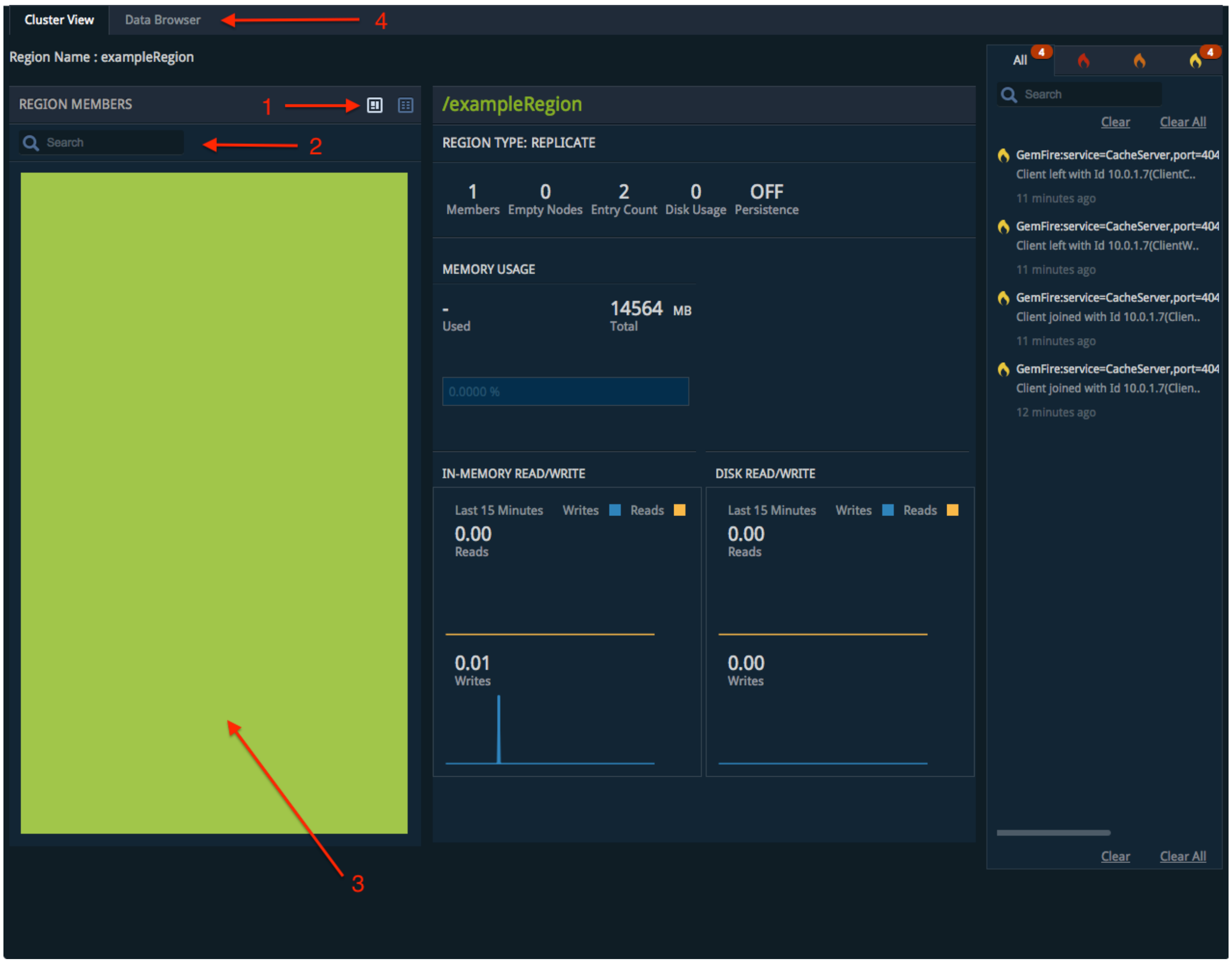
Screen Component	Description
Member Status	Overall status of the member being monitored. Possible statuses include Normal, Warning, or Severe.
Regions	Total number of regions hosted on the member.
Threads	Total number of threads being executed on the member.
Sockets	Total number of sockets currently open on the member.
Load Avg.	Average number of threads on the member that are in the run queue or are waiting for disk I/O over the last minute. Corresponds to the Linux System statistic loadAverage1. If the load average is not available, a negative value is shown.
Clients	Current number of client connections to the member.
Member Regions	Block or table view of the regions hosted on the member.
Regions	When you mouse over a region in block view, a pop-up appears with the following data fields: • *Name*. Region name. • *Type*. For example, REPLICATE, PARTITION. • *EntryCount*. Number of entries in the region. • *EntrySize*. The aggregate entry size (in bytes) of all entries. For replicated regions this field provides a value only if the eviction algorithm is set to EvictionAlgorithm#LRU_ MEMORY. All partition regions can report entry size, but note that the value includes redundant entries and also counts the size of all the secondary entries on the node.

Screen Component	Description
Regions	In table view, the following fields are listed for each region: • *Name*. Region name. • *Type*. For example, REPLICATE, PARTITION. • *EntryCount*. Number of entries in the region. • *EntrySize*. The aggregate entry size (in bytes) of all entries. For replicated regions this field provides a value only if the eviction algorithm is set to EvictionAlgorithm#LRU_ MEMORY. All partition regions can report entry size, but note that the value includes redundant entries and also counts the size of all the secondary entries on the node. • *Scope*. Scope configured for the region. • *Disk Store Name*. Name of disk stores (if any) associated with the region. • *Disk Synchronous*. True if writes to disk are set to synchronous and false if not. This field reflects the configured disk-synchronous region attribute. • *Gateway Enabled*. Whether gateway sender and receiver configurations have been defined on members hosting this region.
Member Clients	In table view, the following fields are listed for each client: • *Id*. DistributedMember ID of the client process. • *Name*. Name of the client process. • *Host*. Hostname or IP address of the client process. • *Connected*. Whether the client process is currently connected to the member. • *Queue Size*. The size of the queue used by server to send events in case of a subscription enabled client or a client that has continuous queries running on the server. • *CPU Usage*. Percentage of CPU being used by the client process. • *Uptime*. Amount of time the client process has been running. • *Threads*. Threads being used by the member clients • *Gets*. Total number of successful get requests completed. • *Puts*. Total number of successful put requests completed.
Key Statistics	Displays a few key performance measurements for the member (over the last 15 minutes).

Screen Component	Description
% CPU Usage	Percentage of CPU used by the member.
Read/Sec	Number of read operations per second that have occurred on the member.
Write/Sec	Number of write operations per second that have occurred on the member. Each put/putAll operation counts as a write; for example, a putAll of 50 entries is counted as one write.
Memory Usage	Total memory used on the member in MB.
No. of JVM Pauses	Number of times the JVM has paused during the last five minutes due to garbage collection or excessive CPU usage.
WAN Information	Displays cluster information. This dialog box only appears if you have configured WAN functionality (gateway senders and gateway receivers).
Disk Throughput	Rate of disk writes on the member.

Region View

The Pulse Region View provides a comprehensive overview of all regions in the cluster:



Region Name : exampleRegion

REGION MEMBERS

/exampleRegion

REGION TYPE: REPLICATE

1 Members 0 Empty Nodes 2 Entry Count 0 Disk Usage OFF Persistence

MEMORY USAGE

- Used 14564 MB Total

0.0000 %

IN-MEMORY READ/WRITE

Last 15 Minutes Writes Reads

0.00 Reads

0.01 Writes

DISK READ/WRITE

Last 15 Minutes Writes Reads

0.00 Reads

0.00 Writes

GemFire:service=CacheServer,port=404
Client left with Id 10.0.1.7(ClientC..
11 minutes ago

GemFire:service=CacheServer,port=404
Client left with Id 10.0.1.7(ClientW..
11 minutes ago

GemFire:service=CacheServer,port=404
Client joined with Id 10.0.1.7(Clien..
11 minutes ago

GemFire:service=CacheServer,port=404
Client joined with Id 10.0.1.7(Clien..
12 minutes ago

Use these basic controls while in Region View:

1. Click the display icons to display all members that host the region using block view or table view. Click the name of a member to change to that member's Member View.
2. Search for specific members that host the current region.
3. Hover over a member name to display information such as the region entry count, entry size, and throughput on that member.
4. Click Cluster View or Data Browser to open those screens.

Region View Screen Components

The following table describes the data elements displayed on the Region View screen.

Screen Component	Description
Region Members	Lists information about VMware GemFire members that host the region, either in block view or table view.
Region Member (Detail View)	When you hover over a region member in block view, a pop-up appears with the following data fields: • *Member Name*. The name of the VMware GemFire member hosting the region. • *EntryCount*. Number of entries for the region on that member. • *EntrySize*. The aggregate entry size (in bytes) of all entries. For replicated regions this field provides a value only if the eviction algorithm is set to EvictionAlgorithm#LRU_ MEMORY. All partition regions can report entry size, but note that the value includes redundant entries and also counts the size of all the secondary entries on the node. • *Accessor*. Indicates whether the member is an accessor member. • *Reads/Writes*. Summary of reads and writes served from memory and from disk stores over the last 15 minutes.
Region Member (Table View)	In table view, the following fields are listed for each region member: • *ID*. The unique member ID. • *Name*. Region name. • *Host*. Member hostname. • *Heap Usage*. The total amount of heap used on the member in MB. • *CPU Usage*. CPU usage as a percent of available CPU. • *Uptime*. The amount of time elapsed since the member started. • *Accessor*. Indicates whether the member is an accessor member.

Screen Component	Description
Region Detail	When you have selected a region, the right hand pane displays the following information about the region: • *Name*. Name of the region. • *Region Path*. Path for the region. • *Type*. For example, REPLICATE, PARTITION • *Members*. Number of members that are hosting the region. • *Empty Nodes*. Nodes where the region DataPolicy is defined as EMPTY or where LocalMaxMemory is set to 0. • *Entry Count*. Total number of entries in the region. • *Disk Usage*. Persistent data usage. • *Persistence*. Indicates whether the region's data is persisted to disk. • *Memory Usage*. The amount of memory used and total available memory (also shown as a percentage). • *Reads/Writes*. Summary of reads and writes served from memory and from disk stores over the last 15 minutes.

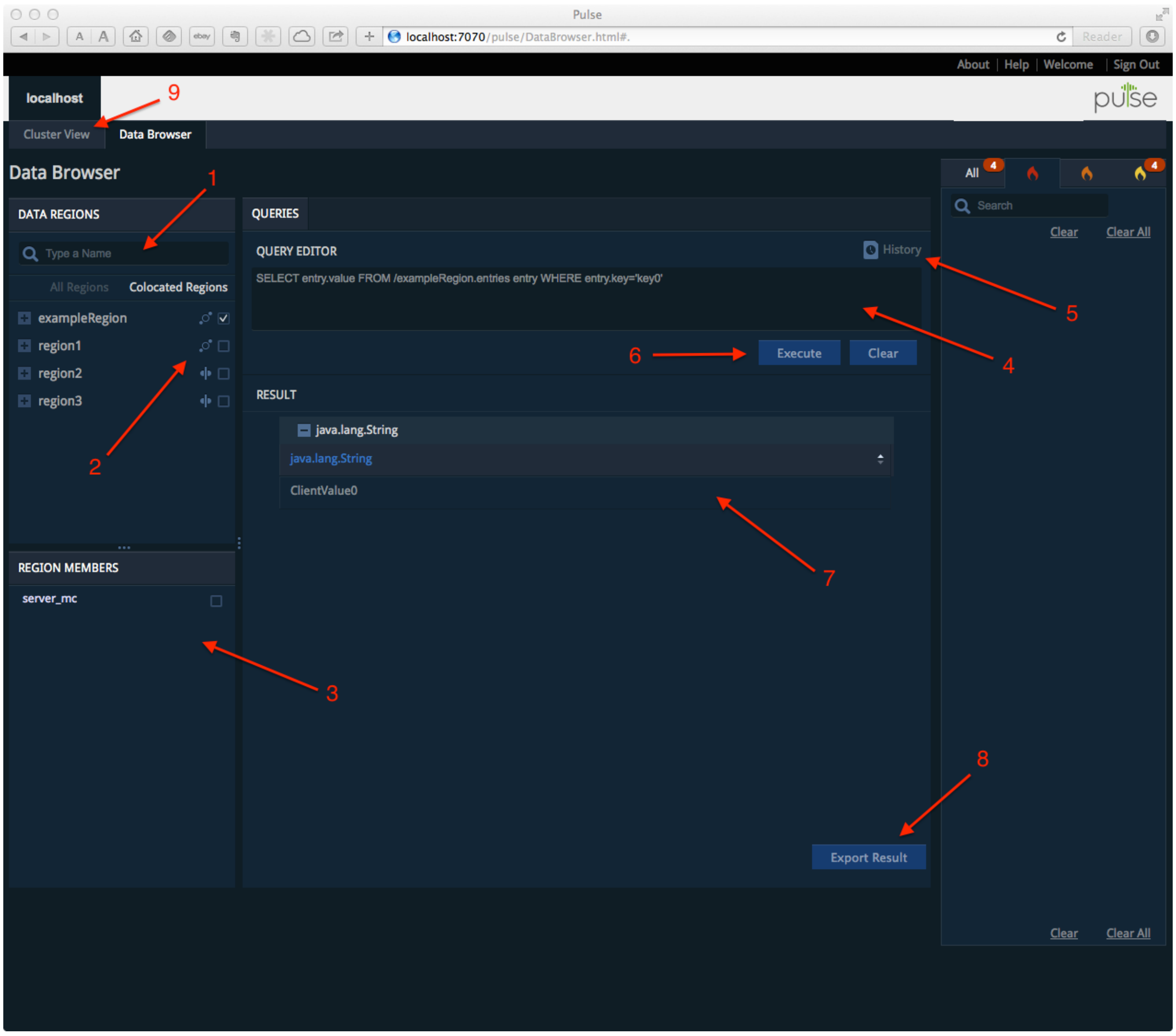
Data Browser

The Pulse Data Browser enables you to query region data. Two key attributes are available on DistributedSystemMXBean. See List of VMware GemFire JMX MBeans that you can use to configure limits for the result sets displayed in Data Browser:

- `QueryResultSetLimit` limits the number of rows that Data Browser queries return. 1000 rows are displayed by default.
- `QueryCollectionsDepth` limits the number of elements of a collection that Data Browser queries return. This attribute applies to query results contain collections such as Map, List, and so forth. The default value is 100 elements.

See the `org.apache.geode.management.DistributedSystemMXBean` JavaDocs for information on available MBean methods and attributes.

The following shows an example Data Browser view:



SELECT entry.value FROM /exampleRegion.entries entry WHERE entry.key='key0'

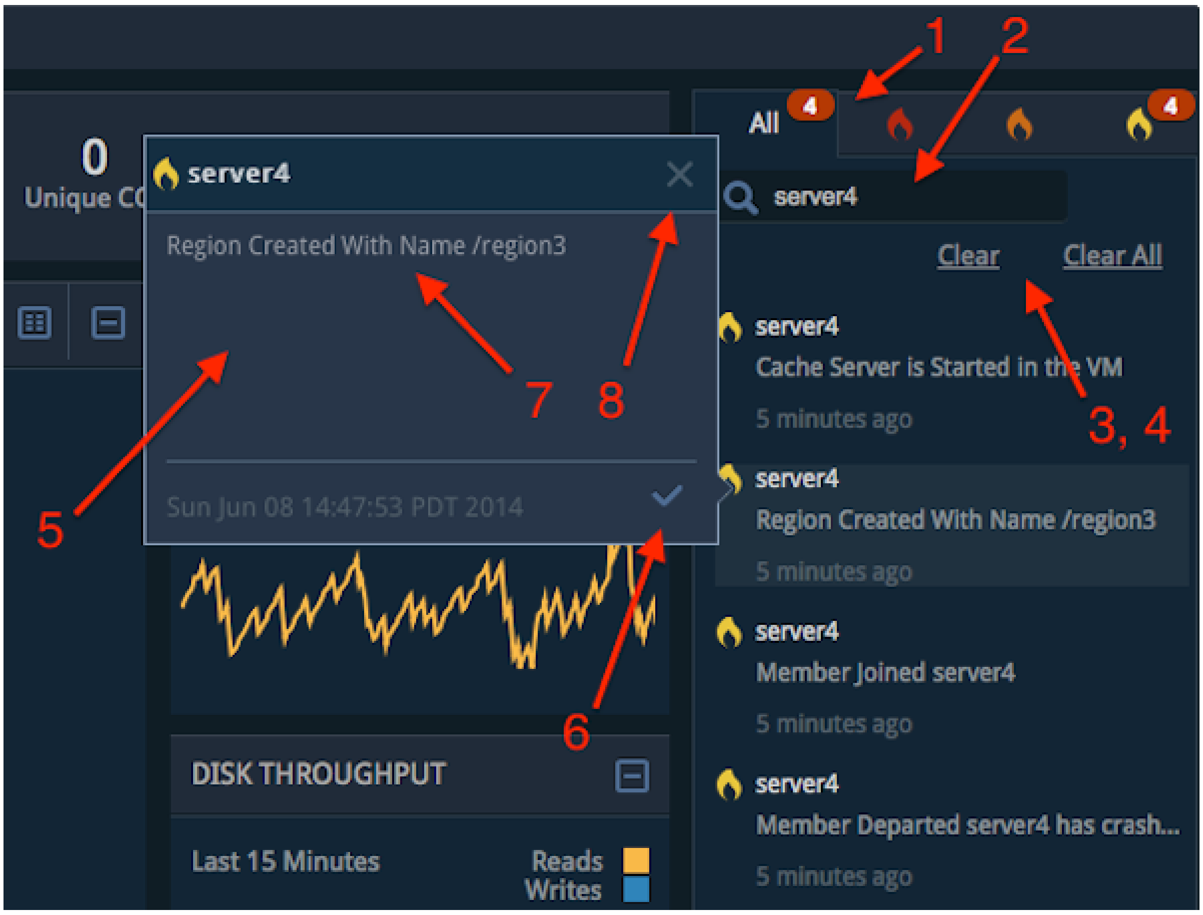
Use these basic controls while in Data Browser view:

1. Search for the name of a specific region.
2. Select one or more regions to display the VMware GemFire members that host those regions. The hosting VMware GemFire members appear in the Region Members section.
3. Select one or more members from the Region Members section to restrict query results to those members.
4. Type in the text of a query to execute. See Querying.
5. Display a list of previously-executed queries. Double-click on a query from the history list to copy it to the Query Editor, or delete the query from your history.
6. Execute your query or clear the contents of the Query Editor.
7. View the current query results.
8. Export the query results to a text file.
9. Return to Cluster View.

Alerts Widget

The Alerts Widget appears in the right portion of the screen and displays a list of alerts.

The alerts displayed for the cluster appear based on the alertLevel field set in the DistributedSystemMXBean. By default, log messages with the level of SEVERE are shown as alerts. You can modify the level by using the `DistributedMXBean.changeAlertLevel` method. See System Alert Notifications for more information.



server4
Cache Server is Started in the VM
5 minutes ago

server4
Region Created With Name /region3
5 minutes ago

server4
Member Joined server4
5 minutes ago

server4
Member Departed server4 has crash...
5 minutes ago

Use these basic controls in the Alerts Widget:

1. Select an alert level to view only alerts with a specific severity.
2. Enter text in the search box to filter the list of alerts.
3. Select an alert and click Clear to remove it from the alert list.
4. Click **Clear All** to remove all alerts from the widget.
5. Double-click an alert to open a pop-up window that displays the full text of the alert message.
6. Click the check mark in an alert pop-up window to acknowledge the alert. Acknowledged alerts display a check mark in the list of alerts.
7. Triple-click the alert in the pop-up or in the alert list to select the message text. You can then copy and paste the text into another application.
8. Click the **X** to close the pop-up alert window.

Visual Statistics Display

The Visual Statistics Display (VSD) utility reads VMware GemFire statistics and produces graphical displays for analysis. VSD is available for download from the VMware Tanzu Network VMware GemFire download page.

- **VSD Overview**

  VSD helps you monitor the performance of VMware GemFire and diagnose performance problems.

- **Installing and Running VSD**

  Start the VSD tool, load statistics files, and maintain the view you want on your statistics.

- **Viewing Statistics in VSD**

  Select statistics and view them using chart templates and customized charts.

- **Quick Guide to Useful Statistics**

  A large number of statistics are intended only for product support and engineering. This topic describes the most important categories and the useful statistics they contain.

VSD System Requirements

VSD works on Linux, macOS, and Windows platforms.

64-Bit Platform Support

VSD is a 32-bit application. If you are running VSD on a 64-bit operating system, you may need to install 32-bit OS libraries to run the application if they are not already installed. On Linux, to find out which libraries are missing you can try running the following:

```
ldd <product_dir>/tools/vsd/bin/vsdwishLinux
```

For 64-bit Windows, you can modify the scripts and executables as described in the note below.

Windows 7 and Later Support

To use VSD on Windows 7, perform the following steps:

1. Start Windows Explorer and navigate to the `GEMFIREPRODUCTDIR\tools\vsd\bin\` directory, where `GEMFIREPRODUCTDIR` corresponds to the location where you installed GemFire.
2. Right click and select properties for vsd.bat.
3. Select the Compatibility tab.
4. Click “Run this program in compatibility mode for” and then select Windows XP SP3.
5. Repeat steps for all the other executables in the `bin` directory.

VSD Overview

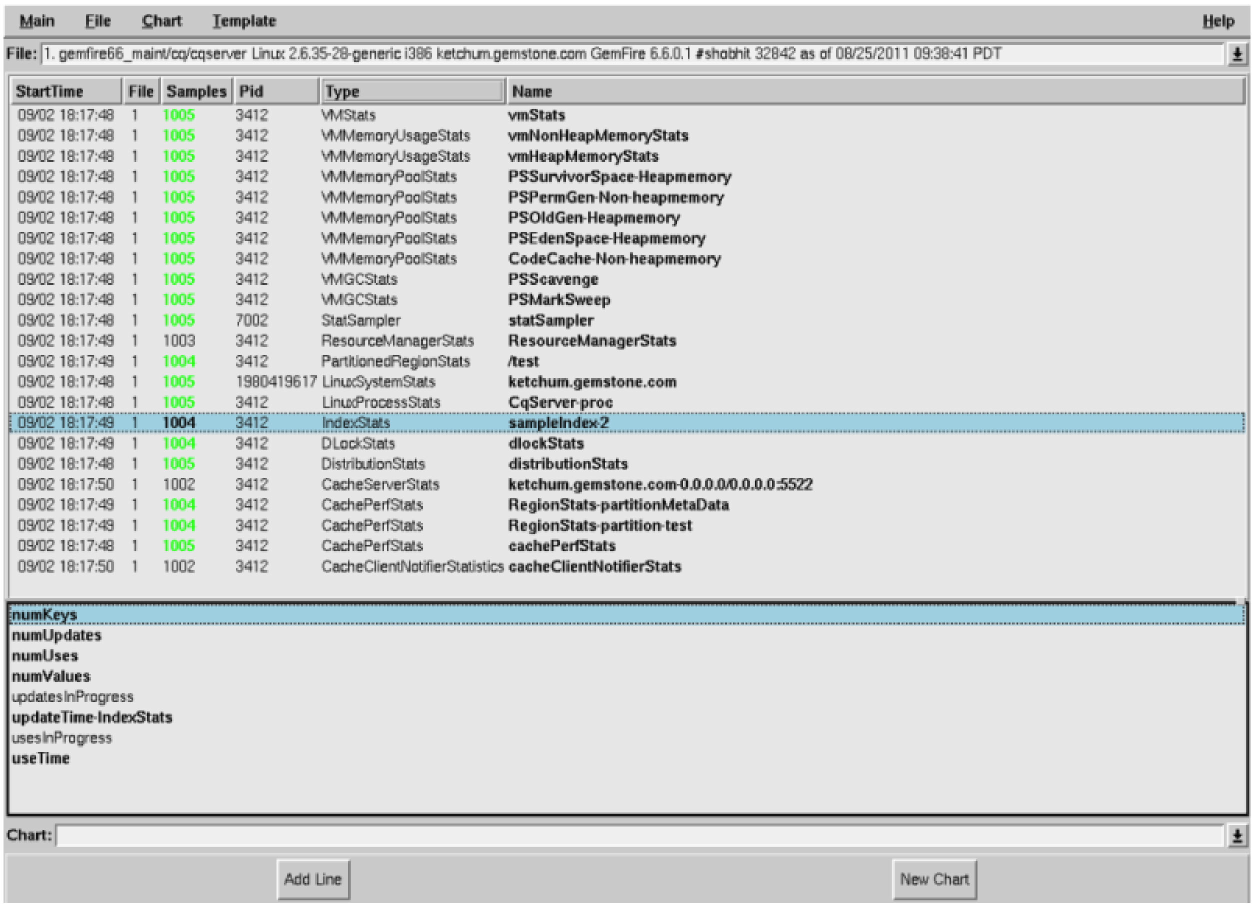
VSD helps you monitor the performance of Pivotal GemFire and diagnose performance problems.

For specific information on the statistics produced by Pivotal GemFire, see Statistics.

Your GemFire product creates a statistical archive file named `filename.gfs`. The file logs useful statistics — counters and gauges that describe the state of the system at a particular moment in time. The file collects statistics at specific sampling intervals, which you can set at various levels to monitor different types of behavior.

The VSD tool reads the sampled statistics and produces graphical displays for analysis. Typically, the points on a line of a VSD graph represent the values for a particular statistic over time. VSD's online help offers more complete reference information about the tool.

The following screenshots of the VSD tool display statistics and a graph analysis of selected statistics.



File: 1. gemfire66_main/cq/cqserver Linux 2.6.35-28-generic i386 ketchum.gemstone.com GemFire 6.6.0.1 #shobhit 32842 as of 08/25/2011 09:38:41 PDT

StartTime	File	Samples	Pid	Type	Name
09/02 18:17:48	1	1005	3412	VMStats	vmStats
09/02 18:17:48	1	1005	3412	VMMemoryUsageStats	vmNonHeapMemoryStats
09/02 18:17:48	1	1005	3412	VMMemoryUsageStats	vmHeapMemoryStats
09/02 18:17:48	1	1005	3412	VMMemoryPoolStats	PSSurvivorSpace-Heapmemory
09/02 18:17:48	1	1005	3412	VMMemoryPoolStats	PSPermGen-Non-heapmemory
09/02 18:17:48	1	1005	3412	VMMemoryPoolStats	PSOldGen-Heapmemory
09/02 18:17:48	1	1005	3412	VMMemoryPoolStats	PSEdenSpace-Heapmemory
09/02 18:17:48	1	1005	3412	VMMemoryPoolStats	CodeCache-Non-heapmemory
09/02 18:17:48	1	1005	3412	VMGCStats	PSScavenge
09/02 18:17:48	1	1005	3412	VMGCStats	PSMarkSweep
09/02 18:17:48	1	1005	7002	StatSampler	statSampler
09/02 18:17:49	1	1003	3412	ResourceManagerStats	ResourceManagerStats
09/02 18:17:49	1	1004	3412	PartitionedRegionStats	/test
09/02 18:17:48	1	1005	1980419617	LinuxSystemStats	ketchum.gemstone.com
09/02 18:17:48	1	1005	3412	LinuxProcessStats	CqServer-proc
09/02 18:17:49	1	1004	3412	IndexStats	sampleIndex-2
09/02 18:17:49	1	1004	3412	DLockStats	dlockStats
09/02 18:17:48	1	1005	3412	DistributionStats	distributionStats
09/02 18:17:50	1	1002	3412	CacheServerStats	ketchum.gemstone.com-0.0.0.0/0.0.0.0:5522
09/02 18:17:49	1	1004	3412	CachePerfStats	RegionStats-partitionMetaData
09/02 18:17:49	1	1004	3412	CachePerfStats	RegionStats-partition-test
09/02 18:17:48	1	1005	3412	CachePerfStats	cachePerfStats
09/02 18:17:50	1	1002	3412	CacheClientNotifierStatistics	cacheClientNotifierStats

numKeys
numUpdates
numUses
numValues
updatesInProgress
updateTime-IndexStats
usesInProgress
useTime

Chart:

:

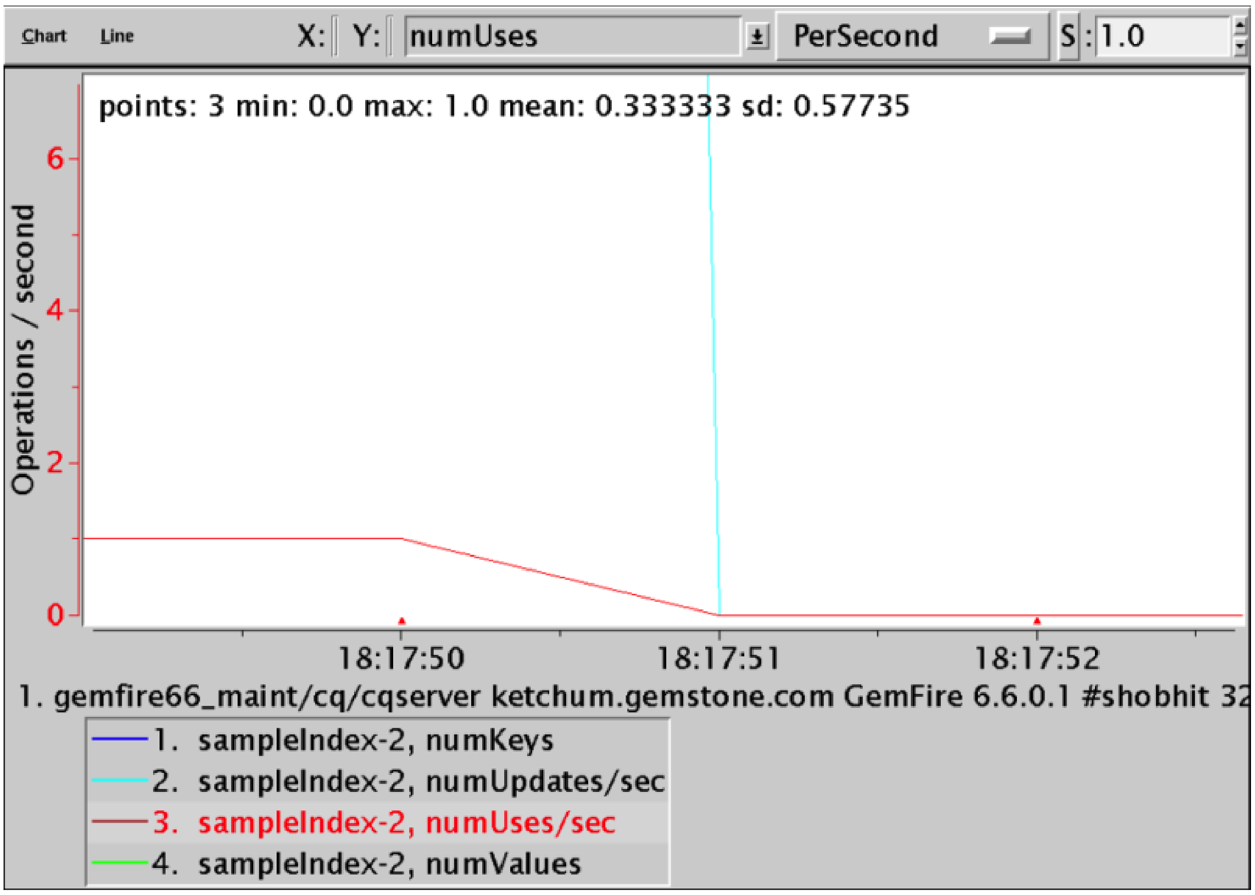


Chart
Line
X:
Y:
numUses
PerSecond
S: 1.0
points: 3 min: 0.0 max: 1.0 mean: 0.333333 sd: 0.57735
Operations / second
6
4
2
0
18:17:50
18:17:51
18:17:52
1. gemfire66_maint/cq/cqserver ketchum.gemstone.com GemFire 6.6.0.1 #shobhit 32
1. sampleIndex-2, numKeys
2. sampleIndex-2, numUpdates/sec
3. sampleIndex-2, numUses/sec
4. sampleIndex-2, numValues

Installing and Running VSD

Start the VSD tool, load statistics files, and maintain the view you want on your statistics.

Install VSD

VSD is a free analysis tool and is provided as-is. See VSD System Requirements to view a list of platforms that are known to work with VSD.

VSD is available for download from the Pivotal GemFire download page. It can be installed anywhere, but if GemFire is installed in *product-dir*, you may wish to install VSD in `product-dir/tools/vsd` so it can be conveniently launched from `gfsh` using the `start vsd` command.

Download the VSD archive, usually named something like `pivotal-gemfire-vsd.zip`, and unpack it in a directory of your choosing. For this example, assume and VSD was downloaded, unzipped, and installed in `product-dir/tools/vsd`.

Note: VSD is a 32-bit application. If you are running VSD on a 64-bit machine, you may need to install 32-bit OS libraries to run the application if they are not already installed. On Linux, to find out which libraries are missing, run the following `ldd` command:

```
ldd <product_dir>/tools/vsd/bin/vsdwishLinux
```

The VSD tool installation has two subdirectories, **bin** and **lib**:

- **bin**. Contains scripts and binaries that can be used to run VSD on a variety of operating systems, including:

    - vsd - script for Solaris, Linux, and Mac
    - vsd.bat - script for Windows
    - vsdwishSunOS - binary for Solaris
    - vsdwishLinux - binary for Linux
    - vsdwishDarwin - binary for Mac
    - vsdwishWindows_NT.exe - binary for Windows
- **lib**. The jars and binary libraries needed to run VSD.

Configure Statistics Sampling in GemFire

Before you use VSD, you must enable the collection of GemFire statistics at runtime. Set the following configurations in gemfire.properties:

```
statistic-sampling-enabled=true
statistic-archive-file=myStats.gfs
```

Since collecting statistics at the default sampling rate of once every second does not affect performance, we recommend that sampling should always be enabled; including during development, testing, and in production.

There is a special category of statistics called time-based statistics that can be very useful in troubleshooting and assessing the performance of some GemFire operations, but they should be used with caution because their collection can affect performance. These statistics can be enabled using the following gemfire.properties configuration:

```
enable-time-statistics=true
```

When the distributed system is up and running, every GemFire instance generates a statistics file. To simplify browsing these statistics in VSD, you may want to copy all the statistics files from all members into one directory so that you can easily load the files into VSD.

Start VSD

To start VSD, you can either execute the scripts directly or start it through the `gfsh` interface. To start VSD using the provided scripts, change directories to `*product-dir*/tools/vsd/bin` and the enter the following command at the prompt:

- **Windows:**

  ```
  prompt>vsd.bat
  ```

  Note: To run VSD on Windows 7 or later, go to the `*product-dir*/tools/vsd/bin` directory. Right-click on `vsd.bat` and select Properties. Click Compatibility and set it to **Windows XP**. Repeat this step for all other executables in the `bin` directory.

- **Linux/Unix, MacOS or Other OS:**

  ```
  $ vsd
  ```

To start VSD using `gfsh`, start a `gfsh` prompt and enter the following command:

```
gfsh>start vsd
```

Load a Statistics File into VSD

You have several options for loading a statistics file into VSD:

- Include the name of one or more statistics files on the VSD command line. Example:

  ```
  vsd <filename.gfs> ...
  ```

- Browse for an existing statistics file through Main > Load Data File.
- Type the full path in the File entry box, then press Enter.
- Switch to a statistics file that you've already loaded by clicking the down-arrow next to the File entry.

After you load the data file, the VSD main window displays a list of entities for which statistics are available. VSD uses color to distinguish between entities that are still running (shown in green) and those that have stopped (shown in black).

Maintain a Current View of the Data File

If you select the menu item File > Auto Update, VSD automatically updates your display, and any associated charts, whenever the data file changes. Alternatively, you can choose File > Update periodically to update the display manually.

About Statistics

Some statistics are cumulative from when the GemFire system was started. Other statistics are instantaneous values that may change in any way between sample collection.

Cumulative statistics are best charted per second or per sample, so that the VSD chart is readable. Absolute values are best charted as No Filter.

.gfs Time Zone Information for Matching Statistics to Log Files

When opening a .gfs file, statistics are shown in the time zone used on the *local computer* where VSD is launched. This can made it harder to relate log files to statistics if the logs are from another time zone.

To open a VSD file with the time zone used when generating it, first you need to know in which time zone the .gfs file is created. To obtain this information, use the following command:

```
strings file.gfs | head
```

For example:

```
$ strings ObjLoader?-31-03.gfs | head
Hongkong
hklp162p.oocl.com
:GemFire? x.x.x
14:46:33 PST
Linux x.x.x
```

After you obtain the time zone, modify your local computer to use the time zone used when obtaining statistics in the .gfs file. For example, on a Mac computer, you can first list available time zones:

```
sudo systemsetup -listtimezones
```

And then export the specific timezone to your environment:

```
export TZ=<timezone>
```

For example, for Hong Kong:

```
export TZ=Asia/Hong_Kong
```

Then use VSD to open the .gfs file that will now display timestamps from the original time zone.

Viewing Statistics in VSD

Select statistics and view them using chart templates and customized charts.

Statistic Levels

Each statistic has a characteristic called a level that reflects the amount of background knowledge that you would need to use the statistic with understanding. You can set up VSD to list (in its main window and in associated charts) only those statistics that are at, or below, a certain level of complexity — common, advanced, or wizard.

To establish the levels of statistics that you want to display in VSD, choose the menu item Main > Statistic Level in the main VSD window.

Select Statistics for Viewing

1. In the VSD list, click the left mouse button to select the entity or entities you want to view.
    - Search for a specific session name or process ID. To find a specific entity, click the mouse in the process list, then press Ctrl-S. When the dialog box appears, enter the PID or name of the entity that you’re looking for. VSD highlights the first entity with that PID or name. To find the next match, press Ctrl-S again. To select the highlighted item, click on it. When you’re done, press Return.
    - Select all entities or by statistic or by type. Statistics are available for various entities, which are listed under the heading Type in the process list.

   Note: You can use the right mouse button to perform these functions:

  - Combine multiple entities into a single line. This can be quite helpful. For example, if you want to measure page reads per second for several hundred entities, you could select all the entities, then combine them into a single line in the chart, thus rendering the data much more readable.
  - Combine multiple entities from different files into a single line.
  - Eliminate flatlines — entities whose values are always zero.
  - Select Single File mode so only one loaded file can be enabled at a time.
  - Select for created lines to have absolute timestamps, which is useful when merging files.
2. Select a statistic for viewing from the statistics list just below the process list.
3. With the selecting statistic, do one of the following:
   - To display the statistic in a new chart, type the name of the chart in the Chart entry box, then click New Chart. (Note: If you don't explicitly specify a chart name, VSD will assign one for you.)
   - To display the statistic in an existing chart, select the chart name in the Chart entry box. Then click on Add Line.
4. To add another statistic to the chart, repeat steps 1 through 3.

Using VSD Chart Templates

VSD templates let you quickly add a set of lines to a chart. Templates are helpful if you find yourself performing the same task frequently in VSD — for example, monitoring the same five or six statistics. By creating a template for the statistics that you want to monitor most frequently, you can automate the task of building charts.

In your template, you can assign a filter for each statistic, to determine how much information is displayed for that statistic. You can also restrict the template to look for extreme conditions (for example, processes that are consuming 90% or more of the CPU).

VSD is shipped with a set of predefined templates, which are maintained in the .vsdtemplates file in your home directory.

Task	Procedure
Create a new chart from a template	In the VSD main window, choose the menu item Template > New Template Chart. This is a good way to display some of the more useful system statistics.
Apply a template to the chart you are viewing	In the Chart window, choose the menu item Chart > Add From Template. (Note: If you have zoomed in on a chart, the template filter is only applied to values within the zoomed range.)
Reread the `.vsdtemplates` template file into VSD after you've edited it	In the VSD main window, choose the menu item Template > Reload Template File.

Task	Procedure
Save the current chart as a template	In the Chart window, configure the chart as you desire, then choose the menu item Chart > Save Template. The template is saved to the `.vsdtemplates` file. If you save the current chart as a template, you may still need to edit the `.vsdtemplates` file so that you can give it a more useful name and make the information and patterns it captures more general.

Chart Menu (Chart Window)

To customize the way VSD displays statistics in your chart, you can choose items from the Chart window's Chart menu.

Chart Menu Item	Effect
Add from Template	Expand template and add resulting lines to chart.
Save Template	Save all lines on chart as a template.
Paste	Paste last item on clipboard.
Print	Print chart.
Snapshot	Write this chart as a graphic to `snapshot.gif`.
Help …	Open Help window.
Zoom In	Zoom in to improve your view of the chart. After you choose this menu item, click to select one corner of the area that you want to zoom. Move the mouse pointer to the opposite corner of the zoom area, then click again. If you have a middle mouse button, you can quickly zoom in on an area by clicking the middle mouse button over it.
Zoom Out	Zoom out by using the menu button or by right-clicking in the chart window.
Compare Two Points	Log information by comparing two points.
Compute Scale All, Unscale All	Adjust the scale of the chart. This helps you view multiple statistics on the same axis.
Show Legend	Display the legend for this chart.
Time Format	Change the format of the time displayed along the X axis.
Show Time Axis Title, Show Left Axis Title, Show Right Axis Title	Display the title alongside the respective axes.
Show Current Values	Display the current X and Y values for the selected line at the top of the chart
Show Min and Max	Display the minimum and maximum values for the selected line at the top of the chart.
Show Line Stats	Display these statistics for the selected line: the number of data samples, the min, max, mean, and standard deviation. The statistics are calculated from all of the data points on the selected line in the region defined by the graph's current X axis. (To change the region, select Zoom In or Zoom Out from the Chart menu.)
Show CrossHairs	Draw cross hairs on graph of item.
Show Grid Lines	Draw grid line on graph of item.
Close	Close chart window.

For additional information about the Chart window, choose Help from the Chart menu.

Line Menu (Chart Window)

Customize your VSD chart display using items from the Line menu (in the Chart window). Line menu commands operate on the currently selected line. To select a line, click on it or on its entry in the chart legend. VSD highlights the selected line.

Line Menu Item	Effect
Log Info	Display a log file showing the line statistics for all data samples in the region defined by the graph's current X axis.
Log Delta	Measure the difference between two values on the selected line. Select the line before choosing this menu item then click on the two points whose difference you want to compute. VSD responds by displaying a log file showing the difference in time and value between the two points; the number of data samples in the selected line segment; and the min, max, mean, and standard deviation of those samples.
Compute Scale	Compute a scale value for the selected line that will make it visible on the current chart. You can also use the Scale entry box to manually change the scale. The default scale value is 1.
Unscale	Reverse the effect of Compute Scale.
Graph on Left Axis	Display the Y axis for the selected line to the left of the chart. Otherwise, the Y axis is displayed to the right. You can use this to view multiple lines on the same chart, by graphing large values on one axis and small values on the opposite axis.
Symbol	Select a new symbol.
Style	Select a line style for connecting points: linear (default), step, natural, quadratic.
Update	Update files used by current line.
Add Lines	Add a line to the current line.
Diff Lines	Remove a line from the current line.
Divide Lines	Divide current line by another line.
Normalize	Normalize current line.
Trim Left, Right	Trim line to left or right of a data point.
Untrim Left, Right	Undo any trim line operations.
Copy	Copy current line to clipboard.
Cut	Cut current line to clipboard.
Delete	Remove the selected line from this chart.

Customizing Your VSD Chart

You can customize and manipulate a VSD chart in many ways:

- To select a line in the chart, click on it or on its entry in the chart legend.
- To delete a line from the chart, click the middle mouse button (if available) on its entry in the chart legend. Alternatively, select the line's entry in the chart legend and choose Line > Delete.
- To find out about a specific point in a chart, hold the mouse pointer over it.

View Statistic Information

To view a description of the most recently selected statistic, along with information about its type, level, and default filter, go to the VSD main window, then choose the menu item Main > Show Statistic Info.

In the Statistic Information window, you can redefine the level and default filter for any VSD statistic.

- The statistic's level — common, advanced, or wizard — allows you to determine whether the statistic is displayed in the VSD statistic list.
- Whenever you add a line to a chart, the filter determines how information is displayed for the selected statistic.

Default Filter	Effect
Default	No Filter if the statistic represents a snapshot of a value. PerSecond if the statistic represents a value that always increases.
No Filter	Display the raw values for the statistic with no filtering.
PerSample	Display the difference between two consecutive samples of the statistic.
PerSecond	Display the difference between two consecutive samples of the statistic, divided by the number of elapsed seconds between the two samples.
Aggregate	Display a running total of per-sample deltas for the statistic. Reset to zero when the delta is zero or changes direction.

Once you have added the line to a chart, you can override its default filter by specifying a new filter from the drop-down menu at the top of the Chart window.

If you leave the Statistic Information window up as you work, it changes to reflect the current statistic. In this way, you can get a quick explanation of any statistic that you're currently examining.

Quick Guide to Useful Statistics

A large number of statistics are intended only for product support and engineering. This topic describes the most important categories and the useful statistics they contain.

For a reference on GemFire statistics, see GemFire Statistics List.

Runtime Configuration

As the name implies, these statistics can help with verifying the runtime configuration of a GemFire system:

- The **number of peer nodes** (i.e. servers or peer accessors) in the system: *DistributionStats:nodes*. This value should be the same for every node in the system.
- The **number of clients and client connections for each server**: *CacheServerStats: currentClients*, and *currentClientConnections*
- The **number of data entries**:
    - *CachePerfStats:entries*. Each region has its own *CachePerfStats* instance per JVM named *RegionStats-REGION-NAME*, or *RegionStats-partition-REGION-NAME* for partitioned regions. Its entries statistic is the number of entries for that region in the JVM.
    - *DiskRegionStatistics* (a per region disk statistic category about the region's disk use): *entriesInVM*, and *entriesOnlyOnDisk* show the number of entries in the JVM (which can also be on disk too), and the number of entries that are only on disk, respectively.
- **Partitioned Region Configuration**: One of the main parameters of Partitioned Region (PR) configuration is the primary bucket distribution. To make sure that primary buckets for a PR are evenly distributed, check the *PartitionedRegionStats.primaryBucketCount* statistic for each partition. This statistic shows the number of primary buckets in a partition.

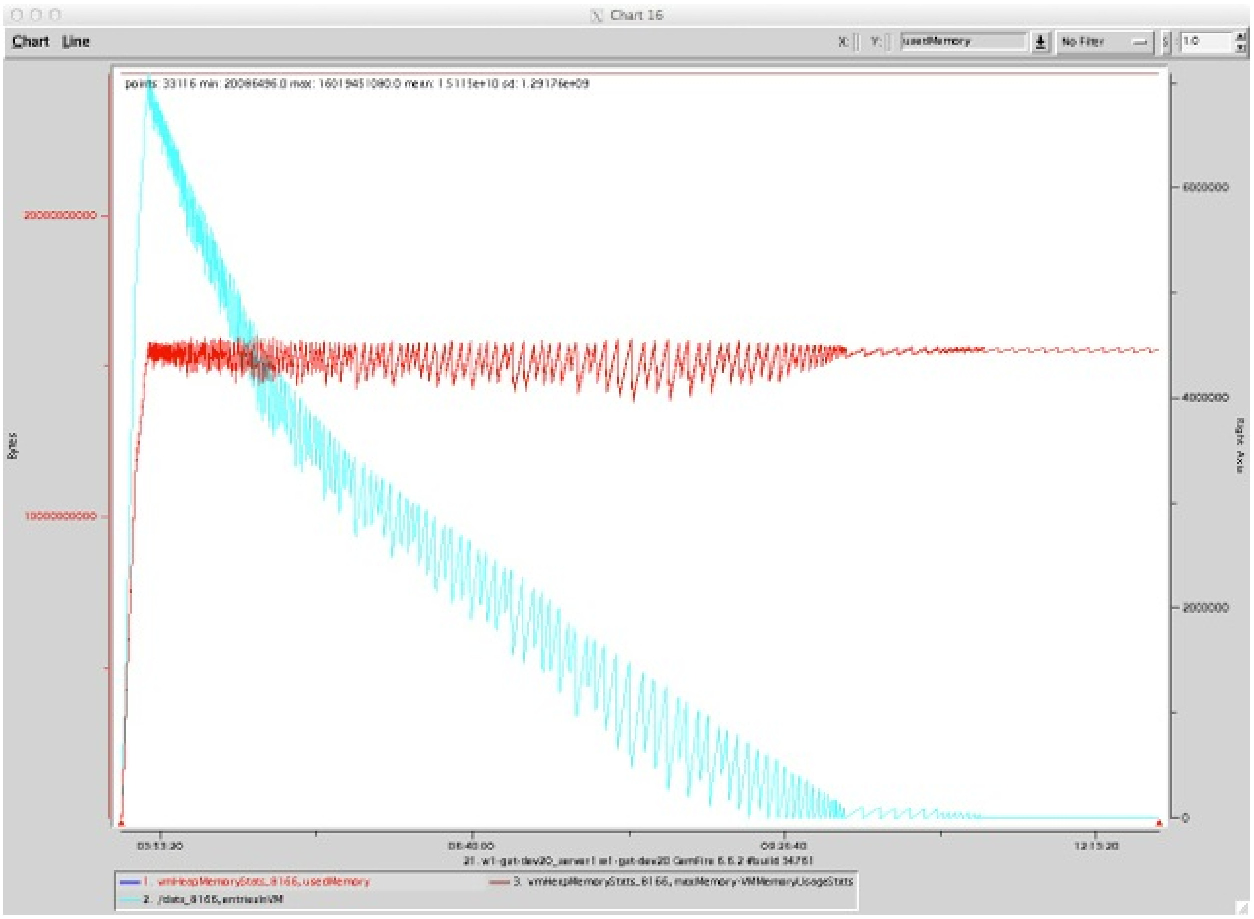
Resources

The resources that are vital for normal operation and performance are memory, file descriptors (most importantly sockets, then files), CPU, network, and disk (when disk operations, such as overflow and persistence, are involved). The following stats cover all those:

- **Memory**: There are several stats categories that show memory usage, for different types and granularity of memory.
    - **Heap**: *VMMemoryUsageStats:vmHeapMemoryStats* are all about heap usage, as are the memory stats under *VMStats:vmStats*: *freeMemory*, *totalMemory*, and *maxMemory*.
    - **Non-heap memory**: *VMMemoryUsageStats:vmNonHeapMemoryStats*.
    - **System-wide memory stats as reported by the OS**: The OS statistic category (e.g. *LinuxSystemStats* on Linux) includes various system level memory statistics, such as *freeMemory*, which shows the free memory on the host (as opposed to related to the JVM process), *physicalMemory* (total physical memory on the host), paging related statistics (*pagesSwappedIn*, *pagesSwappedOut*, *unallocatedSwap*).
    - **Client and gateway queue sizes**: while not actual resources, these queues may be responsible for increased memory usage, so it's good to keep them in mind when investigating memory issues. The client queue stats are in *ClientSubscriptionStats* category: *eventsQueued*, and *eventsRemoved*. The difference between the two is the current queue size. The gateway queue stats are in *GatewaySenderStatistics* category: *eventQueueSize* is the size of the queue.

- **File Descriptors**: file descriptor related statistics are captured in the category *VMStats: fdsOpen* and *fdLimit* show the number of open file descriptors, and the limit on file descriptors for the host, respectively.
- **CPU**: The CPU usage is captured in OS statistic category, e.g. *LinuxSystemStats*. The statistic *cpuActive* shows the percentage of the total available CPU time that has been used in a non-idle state.
- **System load**: OS statistic category (e.g. *LinuxSystemStats*) includes the *loadAverage1*, *loadAverage5*, *loadAverage15* statistics, which show the average system load for 1, 5, and 15 minutes.
- **Network**: OS stats also include network related stats for received (*recv*) and transmitted traffic (*recvBytes*, *xmitBytes*, *recvErrors*, *xmitErrors*).
- **Disk: DiskDirStatistics:diskSpace** shows the amount of disk space used for GemFire disk storage on a given disk. Above mentioned *entriesOnlyOnDisk*, and *entriesInVM* from *DiskRegionStatistics* are useful for determining the distribution of data between memory and disk, for regions that use disk overflow/persistence.

The following chart is an example of examining the *vmHeapMemoryStats* in relation to the *entriesInVM* statistic.



points: 33116 min: 20086496.0 max: 16019451080.0 mean: 1.5115e+10 sd: 1.29176e+09

21: w1-gst-dev20_server1 w1-gst-dev20 GemFire 6.6.2 #build 34761

Throughput for Different Operations

There are several stat categories that capture the throughput for GemFire operations: *CachePerfStats* (non-PR, and PR specific), and*CacheServerStats*, which capture throughput statistics with respect to clients. Note that the PR specific instances of *CachePerfStats* cover only the specific partitioned regions, while the *CachePerfStats* instance includes aggregate stats for all non-PR regions.

- **CachePerfStats** category includes the following stats (all measured in the number of operations per second):
  - *gets*: the number of successful `gets`
  - *puts*: the number of times an entry has been added or replaced as a result of a local operation (put, create, or get which results in a load, netsearch, or netload of a value)
  - *updates*: the number of updates originating remotely
  - *putalls*: the number of `putAll` operations
  - *destroys*: the number of destroys
  - *Function execution: FunctionService*
  - *Queries: queryExecutions*: the number of query executions
  - *Transactions: txCommits*, *txFailures*, *txRollbacks*: the number of successful, failed, and rolled back transactions, respectively
- **CacheServerStats** category includes the following throughput stats for client operations on the cache server:
  - *getRequests, getResponses*
  - *getAllRequests, getAllResponses*
  - *putRequests, putResponses*
  - *putAllRequests, putAllResponses*
  - *queryRequests, queryResponses*
- **Disk operations**: If any disk related statistic categories are present in VSD, it means that there is disk activity (some entries are on disk). Presence of disk operations may explain a drop in throughput, as disk use slows things down.
  - *DiskRegionStatistics* (statistics about a region disk use): *writes, writeTime, writtenBytes, reads, readTime, readBytes*
  - *DiskStoreStatistics* are statistics about a specific disk store’s use of disk. In addition to write/read as those in *DiskRegionStatistics*, this category includes *queueSize* statistic, which shows the current number of entries in the asynchronous queue waiting to be flushed to disk.

VMware GemFire Reference

Reference documents VMware GemFire properties, region attributes, the `cache.xml` file, cache memory requirements, and statistics.

- **gemfire.properties and gfsecurity.properties (VMware GemFire Properties)**

  You use the `gemfire.properties` settings to join a cluster and configure cluster member behavior. Cluster members include applications, the cache server, the locator, and other VMware GemFire processes.

- **cache.xml**

  Use the cache.xml file to set up general cache facilities and behavior and to create and initialize cached data regions. These sections document cache.xml requirements; provide hierarchical diagrams of `<cache>` and `<client-cache>`elements; and describe the function of each element.

- **Region Shortcuts**

  This topic describes the various region shortcuts you can use to configure VMware GemFire regions.

- **Exceptions and System Failures**

  Your application needs to catch certain classes to handle all the exceptions and system failures thrown by VMware GemFire.

- **Memory Requirements for Cached Data**

  VMware GemFire solutions architects need to estimate resource requirements for meeting application performance, scalability and availability goals.

- **VMware GemFire Statistics List**

  This section describes the primary statistics gathered by VMware GemFire when statistics are enabled.

gemfire.properties and gfsecurity.properties: VMware GemFire Properties

You use the `gemfire.properties` settings to join a cluster and configure system member behavior. Cluster members include applications, the cache server, the locator, and other VMware GemFire processes.

You can place any security-related (properties that begin with `security-*`) configuration properties in `gemfire.properties` into a separate `gfsecurity.properties` file. Placing these configuration settings in a separate file allows you to restrict access to security configuration data. This way, you can still allow read or write access for your `gemfire.properties` file.

You can also define provider-specific properties (“ssl” properties) in `gfsecurity.properties` instead of defining them at the command-line or in your environment.

You can specify non-ASCII text in your properties files by using Unicode escape sequences. See Using Non-ASCII Strings in VMware GemFire Property Files for more details.

Note: Unless otherwise indicated, these settings only affect activities within this cluster - not activities between clients and servers or between a gateway sender and gateway receiver in a multi-site installation.

Setting	Definition	Applies to (Server Locator, Client)	Default
ack-severe-alert-threshold	Number of seconds the cluster will wait after the `ack-wait-threshold` for a message to be acknowledged before it issues an alert at severe level. A value of zero disables this feature.	S, L	0
ack-wait-threshold	Number of seconds a distributed message can wait for acknowledgment before it sends an alert to signal that something might be wrong with the system member that is unresponsive. The waiter continues to wait. The alerts are logged in the system member’s log as warnings. Valid values are in the range 0…2147483647	S, L	15
archive-disk-space-limit	Maximum size (in megabytes) of all inactive statistic archive files combined. If this limit is exceeded, inactive archive files are deleted, oldest first, until the total size is within the limit. If set to zero, disk space use is unlimited.	S, L	0
archive-file-size-limit	The maximum size (in megabytes) of a single statistic archive file. Once this limit is exceeded, a new statistic archive file is created, and the current archive file becomes inactive. If set to zero, file size is unlimited.	S, L	0

Setting	Definition	Applies to (Server Locator, Client)	Default
async-distribution-timeout	The number of milliseconds a process that is publishing to this process should attempt to distribute a cache operation before switching over to asynchronous messaging for this process. The switch to asynchronous messaging lasts until this process catches up, departs, or some specified limit is reached, such as async-queue-timeout or async-max-queue-size. To enable asynchronous messaging, the value must be set above zero. Valid values are in the range 0...60000. **Note:** This setting controls only peer-to-peer communication and does not apply to client/server or multi-site communication.	S	0
async-max-queue-size	Affects non-conflated asynchronous queues for members that publish to this member. This is the maximum size the queue can reach (in megabytes) before the publisher asks this member to leave the cluster. Valid values are in the range 0..1024. **Note:** This setting controls only peer-to-peer communication and does not apply to client/server or multi-site communication.	S	8
async-queue-timeout	Affects asynchronous queues for members that publish to this member. This is the maximum milliseconds the publisher should wait with no distribution to this member before it asks this member to leave the cluster. Used for handling slow receivers. **Note:** This setting controls only peer-to-peer communication and does not apply to client/server or multi-site communication.	S, L	60000

Setting	Definition	Applies to (Server Locator, Client)	Default
bind-address	Relevant only for multi-homed hosts - machines with multiple network interface cards. Specifies the adapter card the cache binds to for peer-to-peer communication. Also specifies the default location for VMware GemFire servers to listen on, which is used unless overridden by the `server-bind-address`. An empty string causes the member to listen on the default card for the machine. This is a machine-wide attribute used for system member and client/server communication. It has no effect on locator location, unless the locator is embedded in a member process. Specify the IP address, not the hostname, because each network card may not have a unique hostname. An empty string (the default) causes the member to listen on the default card for the machine.	S, L	*not set*
cache-xml-file	Declarative initialization file for the member's cache.	S	cache.xml
cluster-configuration-dir	This property specifies the directory in which the cluster configuration related disk-store and artifacts are stored. This property is only applicable to dedicated locators that have "enable-cluster-configuration" set to true.	L	*not set*
conflate-events	Used only by clients in a client/server installation. This is a client-side property that is passed to the server. Affects subscription queue conflation in this client's servers. Specifies whether to conflate (true setting), not conflate (false), or to use the server's conflation setting (server).	S	server
conserve-sockets	Specifies whether sockets are shared by the system member's threads. If true, threads share, and a minimum number of sockets are used to connect to the cluster. If false, every application thread has its own sockets for distribution purposes. You can override this setting for individual threads inside your application. WAN deployments increase the messaging demands on a VMware GemFire system. To avoid hangs related to WAN messaging, always use the default setting of `conserve-sockets=false` for VMware GemFire members that participate in a WAN deployment.	S, L	false
delta-propagation	Specifies whether to distribute the deltas for entry updates, instead of the full values, between clients and servers and between peers.	S	true

Setting	Definition	Applies to (Server Locator, Client)	Default
deploy-working-dir	Working directory used when deploying JAR application files to cluster members. This directory can be local and unique to the member or a shared resource. See Deploying Application JARs to VMware GemFire Members for more information.	S	. (current directory)
disable-auto-reconnect	By default, a VMware GemFire member (both locators and servers) will attempt to reconnect and reinitialize the cache after it has been forced out of the cluster by a network partition event or has otherwise been shunned by other members. Use this property to turn off the autoreconnect behavior. See Handling Forced Cache Disconnection Using Autoreconnect for more details.	S, L	false
disable-jmx	By default, VMware GemFire automatically creates JMX MBeans. This boolean, when true, prevents the creation of JMX MBeans. Both gfsh and Pulse require JMX MBeans.	S, L	false
disable-tcp	Boolean indicating whether to disable the use of TCP/IP sockets for inter-cache point-to-point messaging. If disabled, the cache uses datagram (UDP) sockets.	S, L	false
distributed-system-id	Identifier used to distinguish messages from different clusters. This is required for Portable Data eXchange (PDX) data serialization. Set `distributed-system-id` to different values for different systems in a multi-site (WAN) configuration, and to different values for production vs. development environments. This setting must be the same for every member of a given cluster and unique to each cluster within a WAN installation. Valid values are integers in the range -1...255. -1 means no setting.	S, L	-1
durable-client-id	Used only for clients in a client/server installation. If set, this indicates that the client is durable and identifies the client. The ID is used by servers to reestablish any messaging that was interrupted by client downtime.	C	*not set*
durable-client-timeout	Used only for clients in a client/server installation. Number of seconds this client can remain disconnected from its server and have the server continue to accumulate durable events for it.	C	300

Setting	Definition	Applies to (Server Locator, Client)	Default
enable-network-partition-detection	Boolean instructing the system to detect and handle splits in the cluster, typically caused by a partitioning of the network (split brain) where the cluster is running. You must set this property to the same value across all your cluster members. In addition, this property must be set to `true` if you are using persistent regions and configure your regions to use DISTRIBUTED_ACK or GLOBAL scope to avoid potential data conflicts.	S, L	true
enable-cluster-configuration	A value of "true" causes the creation of cluster configuration on dedicated locators. The cluster configuration service on dedicated locator(s) with this property set to "true" would serve the configuration to new members joining the cluster and also save the configuration changes caused by the `gfsh` commands. This property is only applicable to dedicated locators..	L	true
enable-time-statistics	Boolean instructing the system to track time-based statistics for the cluster and caching. Disabled by default for performance reasons and not recommended for production environments. You must also configure `statistic-sampling-enabled` to true and specify a `statistic-archive-file`.	S, L	false
enforce-unique-host	Whether partitioned regions will put redundant copies of the same data in different members running on the same physical machine. By default, VMware GemFire tries to put redundant copies on different machines, but it will put them on the same machine if no other machines are available. Setting this property to true prevents this and requires different machines for redundant copies.	S	false
geode.disallow-internal-messages-without-credentials	A boolean that enables internal message validation when true. Set this system property to true on the `gfsh start server` command line when restarting servers to work with upgraded clients.	S	false
groups	Defines the list of groups that this member belongs to. Use commas to separate group names. Note that anything defined by the roles gemfire property will also be considered a group. See Using Member Groups for more information.	S	*not set*
http-service-bind-address	If set, then the VMware GemFire member binds the embedded HTTP service to the specified address. If this property is not set but the HTTP service is enabled using `http-service-port`, then VMware GemFire binds the HTTP service to the member's local address. Used by the VMware GemFire Pulse Web application and the developer REST API service.	S	*not set*

Setting	Definition	Applies to (Server Locator, Client)	Default
http-service-port	If non-zero, then VMware GemFire starts an embedded HTTP service that listens on this port. The HTTP service is used to host the VMware GemFire Pulse Web application and the development REST API service. If you are hosting the Pulse web app on your own Web server and are not using the development REST API service, then disable this embedded HTTP service by setting this property to zero. Ignored if `jmx-manager` and `start-dev-rest-api` are both set to false.	S	7070
jmx-manager	If true then this member is willing to be a JMX Manager. All the other JMX Manager properties will be used when it does become a manager. If this property is false then all other `jmx-manager-*` properties are ignored.	S, L	false (except on locators)
jmx-manager-bind-address	By default the jmx-manager (when configured with a port) will listen on all the local host's addresses. You can use this property to configure what IP address or host name the JMX Manager will listen on for non-HTTP connections. Ignored if JMX Manager is false or `jmx-manager-port` is zero.	S, L	*not set*
jmx-manager-hostname-for-clients	Lets you control what hostname will be given to clients that ask the locator for the location of a JMX Manager. By default the IP address that the jmx-manager reports is used. But for clients on a different network this property allows you to configure a different hostname that will be given to clients. Ignored if `jmx-manager` is false or `jmx-manager-port` is zero.	S, L	*not set*
jmx-manager-http-port	*Deprecated*. Use `http-service-port` instead.	S, L	7070
jmx-manager-port	The port this JMX Manager will listen to for client connections. If this property is set to zero then VMware GemFire will not allow remote client connections but you can alternatively use the standard system properties supported by the JVM for configuring access from remote JMX clients. Ignored if `jmx-manager` is false.	S, L	1099
jmx-manager-start	If true then this member will start a jmx manager when it creates a cache. Management tools like gfsh can be configured to connect to the jmx-manager. In most cases you should not set this because a jmx manager will automatically be started when needed on a member that sets "jmx-manager" to true. Ignored if jmx-manager is false.	S, L	false

Setting	Definition	Applies to (Server Locator, Client)	Default
jmx-manager-update-rate	The rate, in milliseconds, at which this member will push updates to any JMX Managers. Currently this value should be greater than or equal to the statistic-sample-rate. Setting this value too high will cause stale values to be seen by gfsh and VMware GemFire Pulse.	S, L	2000
load-cluster-configuration-from-dir	Setting this property to "true" causes loading of cluster configuration from "cluster_config" directory in the locator. This property is only applicable to dedicated locators that have "enable-cluster-configuration" set to true.	L	false
locator-wait-time	The number of seconds that a member should wait for a locator to start if a locator is not available when attempting to join the cluster. Use this setting when you are starting locators and peers all at once. This timeout allows peers to wait for the locators to finish starting up before attempting to join the cluster.	S	0

Setting	Definition	Applies to (Server Locator, Client)	Default

locators	The list of locators used by system members. The list must be configured consistently for every member of the cluster. If the list is empty, locators are not used.
	
	For each locator, provide a host name and/or address (separated by '@', if you use both), followed by a port number in brackets. Examples:
	
	`locators=addr1[port1],addr2[port2]`
	
	`locators=host1@addr1[port1]`
	
	`locators=host1[port1],host2[port2]`
	
	Note:
	On multi-homed hosts, this last notation will use the default address. If you use bind addresses for your locators, explicitly specify the addresses in the locators list—do not use just the hostname.
	
	If you have values specified for the `locators` property, the `mcast-port` property defaults to 0.
	
	Note:
	If you specify invalid DNS hostnames in this property, any locators or servers started with `gfsh` will not produce log files. Make sure you provide valid DNS hostnames before starting the locator or server with `gfsh`.
		S, L	*not set*

lock-memory	When `true`, locks heap and off-heap memory into RAM to prevent the operating system from paging the memory out to disk.	S, L	false
log-disk-space-limit	Maximum size in megabytes of all inactive log files combined. If this limit is exceeded, inactive log files are deleted, oldest first, until the total size is within the limit. If set to zero, disk space use is unlimited.	S, L	0

Setting	Definition	Applies to (Server Locator, Client)	Default
log-file	File to which a running system member writes log messages. If set to null, the default is used. Each member type has its own default output: • application: standard out • locator: `<locator_name>.log` • server: `<server_name>.log`	S, L	null
log-file-size-limit	Maximum size in megabytes of a log file before it is closed and logging rolls on to a new (child) log file. If set to 0, log rolling is disabled.	S, L	0
log-level	Level of detail of the messages written to the system member's log. Setting log-level to one of the ordered levels causes all messages of that level and greater severity to be printed. Valid values from lowest to highest are fine, config, info, warning, error, severe, and none.	S, L	config
max-wait-time-reconnect	Maximum number of milliseconds to wait for the cluster to reconnect on each reconnect attempt.	S, L	60000
mcast-address	Address used to discover other members of the cluster. Only used if mcast-port is non-zero. This attribute must be consistent across the cluster. Select different multicast addresses and different ports for different clusters. Do not just use different addresses. Some operating systems may not keep communication separate between systems that use unique addresses but the same port number. This default multicast address was assigned by IANA (multicast-addresses). Consult the IANA chart when selecting another multicast address to use with VMware GemFire. **Note:** This setting controls only peer-to-peer communication and does not apply to client/server or multi-site communication. If multicast is enabled, distributed regions use it for most communication. Partitioned regions only use multicast for a few purposes, and mainly use either TCP or UDP unicast.	S, L	239.192.81.1 for IPv4 (the default IP version) FF38::1234 for IPv6

Setting	Definition	Applies to (Server Locator, Client)	Default
mcast-flow-control	Tuning property for flow-of-control protocol for unicast and multicast no-ack UDP messaging. Compound property made up of three settings separated by commas: byteAllowance, rechargeThreshold, and rechargeBlockMs. Valid values range from these minimums: 10000,0.1,500 to these maximums: no_maximum ,0.5,60000. **Note:** This setting controls only peer-to-peer communication, generally between distributed regions.	S, L	1048576, 0.25, 5000
mcast-port	Port used, along with the mcast-address, for multicast communication with other members of the cluster. If zero, multicast is disabled. Valid values are in the range 0..65535. **Note:** Select different multicast addresses and ports for different clusters. Do not just use different addresses. Some operating systems may not keep communication separate between systems that use unique addresses but the same port number. **Note:** This setting controls only peer-to-peer communication and does not apply to client/server or multi-site communication.	S, L	0

Setting	Definition	Applies to (Server Locator, Client)	Default
mcast-recv-buffer-size	Size of the socket buffer used for incoming multicast transmissions. You should set this high if there will be high volumes of messages. Valid values are in the range 2048.. OS_maximum. **Note:** The default setting is higher than the default OS maximum buffer size on Unix, which should be increased to at least 1 megabyte to provide high-volume messaging on Unix systems. **Note:** This setting controls only peer-to-peer communication and does not apply to client/server or multi-site communication.	S, L	1048576
mcast-send-buffer-size	The size of the socket buffer used for outgoing multicast transmissions. Valid values are in the range 2048.. OS_maximum. **Note:** This setting controls only peer-to-peer communication and does not apply to client/server or multi-site communication.	S, L	65535
mcast-ttl	How far multicast messaging goes in your network. Lower settings may improve system performance. A setting of 0 constrains multicast messaging to the machine. **Note:** This setting controls only peer-to-peer communication and does not apply to client/server or multi-site communication.	S, L	32

Setting	Definition	Applies to (Server Locator, Client)	Default
member-timeout	VMware GemFire uses the `member-timeout` server configuration, specified in milliseconds, to detect unresponsive members. VMware GemFire members use a combination of UDP heartbeat messages and TCP sockets to monitor the health of other members. If a member is suspected to be unreachable, two active checks of the member are performed. Each of those checks waits a maximum of one `member-timeout`. If those health checks receive a connection failure or no response within the timeout, the member is considered dead and is removed from the list of members. This setting also controls how frequently heartbeat messages are sent. They are sent at a frequency of half the `member-timeout`. The total time it takes to remove a member that is not responding to any network traffic is therefore 2 to 3 times the `member-timeout`. This is not the minimum time it takes to remove a member from the list of members in the cluster. If the Java process has crashed, but the operating system can still return a connection failure response to the health checks, the crashed member may be removed from the membership list immediately. Valid values are in the range 1000..600000.	S, L	5000
membership-port-range	The range of ports available for unicast UDP messaging and for TCP failure detection. This is specified as two integers separated by a hyphen. Different members can use different ranges. VMware GemFire randomly chooses at least two unique integers from this range for the member, one for UDP unicast messaging and the other for TCP failure detection messaging. If tcp-port is configured to 0, it will also randomly select a port from this range for TCP sockets used for peer-to-peer communication only. Therefore, the specified range must include at least three available port numbers (UDP, FD_SOCK, and TCP DirectChannel). The system uniquely identifies the member using the combined host IP address and UDP port number. You may want to restrict the range of ports that VMware GemFire uses so the product can run in an environment where routers only allow traffic on certain ports.	S, L	41000-61000
memcached-port	If specified and is non-zero, sets the port number for an embedded Gemcached server and starts the Gemcached server.	S	0

Setting	Definition	Applies to (Server Locator, Client)	Default
memcached-protocol	Sets the protocol used by an embedded Gemcached server. Valid values are `BINARY` and `ASCII.` If you omit this property, the ASCII protocol is used.	S	ASCII
name	Symbolic name used to identify this system member.	S, L	*not set*

off-heap-memory-size	Specifies the size of off-heap memory in megabytes (m) or gigabytes (g). For example:
	`off-heap-memory-size=4096m`
	`off-heap-memory-size=120g`
		S	*not set*

redundancy-zone	Defines this member's redundancy zone. Used to separate member's into different groups for satisfying partitioned region redundancy. If this property is set, VMware GemFire will not put redundant copies of data in members with the same redundancy zone setting. See Configure High Availability for a Partitioned Region for more details.	S	*not set*

remote-locators	Used to configure the locators that a cluster will use in order to connect to a remote site in a multi-site (WAN) configuration. To use locators in a WAN configuration, you must specify a unique distributed system ID (`distributed-system-id`) for the local cluster and remote locator(s) for the remote clusters to which you will connect.
	For each remote locator, provide a host name and/or address (separated by '@', if you use both), followed by a port number in brackets. Examples:
	`remote-locators=addr1[port1],addr2[port2]`
	`remote-locators=host1@addr1[port1]`
	`remote-locators=host1[port1],host2[port2]`
		L	*not set*

remove-unresponsive-client	When this property is set to true, the primary server drops unresponsive clients from all secondaries and itself. Clients are deemed unresponsive when their messaging queues become full on the server. While a client's queue is full, puts that would add to the queue block on the server.	S	false

Setting	Definition	Applies to (Server Locator, Client)	Default
security-*	Any security-related (properties that begin with `security-`) configuration properties that are normally configured in `gemfire.properties` can be moved to a separate `gfsecurity.properties` file. Placing these configuration settings in a separate file allows you to restrict access to security configuration data. This way, you can still allow read or write access for your `gemfire.properties` file.	S, L	*not set*
security-client-accessor	**Deprecated.** Used for authorization. Static creation method returning an `AccessControl` object, which determines authorization of client-server cache operations. This specifies the callback that should be invoked in the pre-operation phase, which is when the request for the operation is received from the client.	S, L	*not set*
security-client-accessor-pp	**Deprecated.** Used for authorization. The callback that should be invoked in the post-operation phase, which is when the operation has completed on the server but before the result is sent to the client. The post-operation callback is also invoked for the updates that are sent from server to client through the notification channel.	S, L	*not set*
security-client-auth-init	Used for authentication. Defines the fully-qualified static method that instantiates and returns the `AuthInitialize` object. `AuthInitialize` is responsible for producing the necessary credentials for client authentication.	S, L	*not set*
security-client-authenticator	**Deprecated.** Used for authentication. Static creation method returning an `Authenticator` object, which is used by a peer to verify the credentials of the connecting peer.	S, L	*not set*
security-client-dhalgo	**Deprecated.** Use `ssl-enabled-components` instead. Used for authentication. For secure transmission of sensitive credentials like passwords, you can encrypt the credentials using the Diffie-Hellman key-exchange algorithm. Do this by setting the security-client-dhalgo system property on the clients to the name of a valid, symmetric key cipher supported by the JDK.	S, L	*not set*
security-log-file	Used with authentication. The log file for security log messages. If not specified, the member's regular log file is used.	S, L	*not set*
security-log-level	Used with authentication. Logging level detail for security log messages. Valid values from lowest to highest are fine, config, info, warning, error, severe, and none.	S, L	config

Setting	Definition	Applies to (Server Locator, Client)	Default
security-manager	Specifies the implementation of the `SecurityManager` interface that implements the callbacks that do authentication and authorization.	S, L	*not set*
security-peer-auth-init	Used for authentication. Defines the fully-qualified static method that instantiates and returns the `AuthInitialize` object. `AuthInitialize` is responsible for producing the necessary credentials for peer authentication.	S, L	*not set*
security-peer-authenticator	**Deprecated.** Used with authentication. Static creation method returning an `Authenticator` object, which is used by a peer to verify the credentials of the connecting peer.	S, L	*not set*
security-peer-verifymember-timeout	Used with authentication. Timeout in milliseconds used by a peer to verify membership of an unknown authenticated peer requesting a secure connection.	S, L	1000
security-post-processor	Specifies the implementation of the `PostProcessor` interface that implements user-defined callbacks that can change the returned results of region get operations.	S, L	*not set*
security-udp-dhalgo	Specifies a string that defines the name of a valid, symmetric key cipher supported by the JDK. When defined, the named cipher will be used for server-to-server UDP communications.	S	*not set*
serializable-object-filter	A semicolon-separated list of items that become full class names of objects that the system will serialize when the property validate-serializable-objects is set to true. The list is expanded using the patterns specified in the `createFilter` method at ObjectInputFilter.Config.html.	S, C	"!*"

Setting	Definition	Applies to (Server Locator, Client)	Default
server-bind-address	Relevant only for multi-homed hosts - machines with multiple network interface cards. Network adapter card a VMware GemFire server binds to for client/server communication. You can use this to separate the server's client/server communication from its peer-to-peer communication, spreading the traffic load. This is a machine-wide attribute used for communication with clients in client/server and multi-site installations. This setting has no effect on locator configuration. Specify the IP address, not the hostname, because each network card may not have a unique hostname. An empty string causes the servers to listen on the same card used for peer-to-peer communication. This is either the `bind-address` or, if that is not set, the machine's default card.	S	*not set*
socket-buffer-size	Receive buffer sizes in bytes of the TCP/IP connections used for data transmission. To minimize the buffer size allocation needed for distributing large, serializable messages, the messages are sent in chunks. This setting determines the size of the chunks. Larger buffers can handle large messages more quickly, but take up more memory.	S, L	32768
socket-lease-time	Time, in milliseconds, a thread can have exclusive access to a socket it is not actively using. A value of zero causes socket leases to never expire. This property is ignored if conserve-sockets is true. Valid values are in the range 0..600000.	S, L	60000
ssl-enabled-components	Components for which to enable SSL. Comma-separated list of one or more of (cluster, gateway, jmx, locator, server, web) or "all". When defining this property, also set `security-udp-dhalgo` to secure UDP cluster communications.	C, S, L	all
ssl-endpoint-identification-enabled	Boolean. When set to true, causes clients to validate the server's hostname using the server's certificate.	C, S, L	false
ssl-use-default-context	Boolean. When set to true, allows VMware GemFire to use the default SSL context. When enabled, also sets ssl-endpoint-identification-enabled to true.	C, S, L	false
ssl-require-authentication	Boolean. Require two-way authentication for SSL-enabled components. Applies to all components except web.	S, L	true

Setting	Definition	Applies to (Server Locator, Client)	Default
ssl-http-require-authentication	Boolean. Require two-way authentication for web component.	S, L	false
ssl-default-alias	String. Default certificate name. If empty, use first certificate in key store.	S, L	
ssl-*component*-alias	String. Certificate name for specified *component*, which is one of: cluster, gateway, jmx, locator, server, or web.	S, L	
ssl-ciphers	Comma-separated list of SSL ciphers or "any"	S, L	any
ssl-parameter-extension	Specifies the implementation of the `SSLParameterExtension` interface that implements the SSL parameter extensions.	S, L	*not set*
ssl-protocols	Comma-separated list of SSL protocols or "any"	S, L	any
ssl-keystore, ssl-keystore-password	Strings. Path to key store, key store password.	S, L	
ssl-truststore, ssl-truststore-password	Strings. Path to trust store, trust store password.	S, L	
ssl-keystore-type, ssl-truststore-type	Strings. Type of key store or trust store. "JKS" indicates Java. One common alternative is "pkcs12".	S, L	JKS
start-dev-rest-api	If set to true, then the developer REST API service will be started when cache is created. REST service can be configured using `http-service-port` and `http-service-bind-address` properties.	S	false

start-locator	If set, automatically starts a locator in the current process when the member connects to the cluster and stops the locator when the member disconnects.
	
	To use, specify the locator with an optional address or host specification and a required port number, in one of these formats:
	
	`start-locator=address[port1]`
	
	`start-locator=port1`
	
	If you only specify the port, the address assigned to the member is used for the locator.
	
	If not already there, this locator is automatically added to the list of locators in this set of `gemfire properties`.
		S	*not set*

Setting	Definition	Applies to (Server Locator, Client)	Default
statistic-archive-file	The file to which the running system member writes statistic samples. For example: "StatisticsArchiveFile.gfs". An empty string disables archiving. Adding .gz suffix to the file name causes it to be compressed.	S, L	*not set*
statistic-sample-rate	How often to sample statistics, in milliseconds. Valid values are in the range 100..60000.	S, L	1000
statistic-sampling-enabled	Whether to collect and archive statistics on the member. Statistics sampling provides valuable information for ongoing system tuning and troubleshooting purposes. Sampling statistics at the default sample rate does not impact system performance. We recommend enabling statistics sampling in production environments. **Note:** This setting does not apply to partitioned regions, where statistics are always enabled.	S, L	false
tcp-port	The TCP port to listen on for cache communications. If set to zero, the operating system selects an available port. Each process on a machine must have its own TCP port. Note that some operating systems restrict the range of ports usable by non-privileged users, and using restricted port numbers can cause runtime errors in VMware GemFire startup. Valid values are in the range 0..65535.	S, L	0
thread-monitor-enabled	Boolean. When true, enables monitoring of VMware GemFire-created operational threads. Informational messages are written to the log file.	S	true
thread-monitor-interval-ms	The time interval (in milliseconds) with which thread monitoring is scheduled to run.	S	60000
thread-monitor-time-limit-ms	The time period (in milliseconds) after which the monitored thread is considered to be stuck.	S	30000

Setting	Definition	Applies to (Server Locator, Client)	Default
tombstone-gc-threshold	The number of tombstones that can accumulate before the VMware GemFire member triggers garbage collection for tombstones. See How Destroy and Clear Operations Are Resolved.	S	100000
udp-fragment-size	Maximum fragment size, in bytes, for transmission over UDP unicast or multicast sockets. Smaller messages are combined, if possible, for transmission up to the fragment size setting. Valid values are in the range 1000..60000.	S, L	60000
udp-recv-buffer-size	The size of the socket buffer used for incoming UDP point-to-point transmissions. If disable-tcp is false, a reduced buffer size of 65535 is used by default. The default setting of 1048576 is higher than the default OS maximum buffer size on Unix, which should be increased to at least 1 megabyte to provide high-volume messaging on Unix systems. Valid values are in the range 2048.. OS_maximum.	S, L	1048576
udp-send-buffer-size	The size of the socket buffer used for outgoing UDP point-to-point transmissions. Valid values are in the range 2048..OS_maximum.	S, L	65535
use-cluster-configuration	This property is only applicable for data members (non-client and non-locator). A value of "true" causes a member to request and use the configuration from cluster configuration services running on dedicated locators. Setting this property to "false" causes a member to not request the configuration from the configuration services running on the locator(s).	S	true
user-command-packages	A comma separated list of Java packages that contain classes implementing the `CommandMarker` interface. Matching classes will be loaded when the VM starts and will be available in the GFSH command-line utility.	S	*not set*
validate-serializable-objects	A boolean that defaults to false. When true, instances of classes that are not internal to VMware GemFire and whose class name is not allowed by the list defined in the serializable-object-filter property will not be permitted to be deserialized. An `IncompatibleClassException` is thrown for objects not listed. JDK 8 build 121 or a later build must be installed to use this property. Servers and clients that do not meet this requirement will throw an exception upon startup.	S, C	false

- **Using Non-ASCII Strings in VMware GemFire Property Files**

You can specify Unicode (non-ASCII) characters in VMware GemFire property files by using a `\uXXXX` escape sequence.

Using Non-ASCII Strings in VMware GemFire Property Files

You can specify Unicode (non-ASCII) characters in VMware GemFire property files by using a `\uXXXX` escape sequence.

For a supplementary character, you need two escape sequences, one for each of the two UTF-16 code units. The XXXX denotes the 4 hexadecimal digits for the value of the UTF-16 code unit. For example, a properties file might have the following entries:

```
s1=hello there
s2=\u3053\u3093\u306b\u3061\u306f
```

For example, in `gemfire.properties`, you might write:

```
log-file=my\u00df.log
```

to indicate the desired propery definition of `log-file=myß.log`.

If you have edited and saved the file in a non-ASCII encoding, you can convert it to ASCII with the `native2ascii` tool included in your Oracle Java distribution. For example, you might want to do this when editing a properties file in Shift_JIS, a popular Japanese encoding.

For more information on internationalization in Java, see http://www.oracle.com/technetwork/java/javase/tech/intl-139810.html.

cache.xml

Use the cache.xml file to set up general cache facilities and behavior and to create and initialize cached data regions. These sections document cache.xml requirements; provide hierarchical diagrams of `<cache>` and `<client-cache>`elements; and describe the function of each element.

Note: You can configure most elements of the cache.xml file and apply it to your entire cluster by using the gfsh and cluster configuration service. See gfsh Limitations for a list of items you cannot configure in gfsh and must still configure in cache.xml.

- **cache.xml Quick Reference**

  This section documents cache.xml file requirements and variables. It also points you to specific element sections for server, client, and WAN configuration.

- **<cache> Element Hierarchy**

  This section shows the hierarchy of `<cache>` element sub-elements that you use to configure VMware GemFire caches and servers.

- **<cache> Element Reference**

  This section documents the `cache.xml` sub-elements used for VMware GemFire server configuration. All elements are sub-elements of the `<cache>` element.

- **<client-cache> Element Hierarchy**

This section shows the hierarchy of `<client-cache>` element sub-elements that you use to configure VMware GemFire caches and clients.

- **<client-cache> Element Reference**

  This section documents all `cache.xml` elements that you use to configure VMware GemFire clients. All elements are sub-elements of the `<client-cache>` element.

cache.xml Quick Reference

This section documents cache.xml file requirements and variables. It also points you to specific element sections for server, client, and WAN configuration.

- Cache XML Requirements
- Variables in cache.xml
- Configuration Quick Reference

Cache XML Requirements

The cache.xml file has these requirements:

- The contents must conform to the XML schema definition provided in cache-1.0.xsd. The schema definition file is available at http://geode.apache.org/schema/cache/cache-1.0.xsd.
- The file must include a <cache> schema declaration of one of the following forms:
  - Server or peer cache:

    ```
    <?xml version="1.0" encoding="UTF-8"?>
    <cache
        xmlns="http://geode.apache.org/schema/cache"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://geode.apache.org/schema/cache http://geod
    e.apache.org/schema/cache/cache-1.0.xsd"
        version="1.0">
    ...
    </cache>
    ```

  - Client cache:

    ```
    <?xml version="1.0" encoding="UTF-8"?>
    <client-cache
        xmlns="http://geode.apache.org/schema/cache"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://geode.apache.org/schema/cache http://geod
    e.apache.org/schema/cache/cache-1.0.xsd"
        version="1.0">
    ...
    </client-cache>
    ```

- Any class name specified in the file **must have a public zero-argument constructor** and must implement the `org.apache.geode.cache.Declarable` interface. Parameters declared in the XML for the class are passed to the class init method.

Variables in cache.xml

You can use variables in the `cache.xml` to customize your settings without modifying the XML file.

Set your variables in Java system properties when you start your cache server or application process.

Example cache.xml with variables and the gfsh `start server` command that sets the variables:

```
<?xml version="1.0" encoding="UTF-8"?>
<cache
    xmlns="http://geode.apache.org/schema/cache"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://geode.apache.org/schema/cache http://geode.apache.org/s
chema/cache/cache-1.0.xsd"
    version="1.0">
  <cache-server port="${PORT}" max-connections="${MAXCNXS}"/>
  <region name="root">
    <region-attributes refid="REPLICATE"/>
  </region>
</cache>
```

```
gfsh>start server --name=server2 --cache-xml-file=cache.xml --J=-DPORT=30333 --J=-DMAX
CNXS=77
```

Configuration Quick Reference

To configure cache servers, clients, and WAN topologies, see the following sections:

- Server Configuration
  - <cache> Element Reference
  - <cache-server>
  - <region>
  - <region-attributes>

  You can set the same server configuration properties using the `org.apache.geode.cache.server.CacheServer` and `org.apache.geode.cache.Cache` interfaces. For detailed information, see the online Java API documentation.
- Client Configuration
  - <client-cache> Element Reference
  - <pool>
  - <region>

  You can set the same client configuration properties using the `org.apache.geode.cache.clientClientCache` and `Pool` interfaces. For detailed information, see the online Java API documentation.
- Multi-site (WAN) Configuration and Asynchronous Event Queue Configuration
  - <gateway-sender>

- <gateway-receiver>
- <async-event-queue>

The gateway sender and receiver APIs in `org.apache.geode.cache.util` provide corresponding getter and setter methods for these attributes.

<cache> Element Hierarchy

This section shows the hierarchy of `<cache>` element sub-elements that you use to configure VMware GemFire caches and servers.

For details, see <cache> Element Reference.

```
<cache>
   <cache-transaction-manager>
      <transaction-listeners>
      <transaction-writer>
   <gateway-hub>
      <gateway>
         <gateway-endpoint>
         <gateway-listener>
         <gateway-queue>
   <gateway-sender>
      <gateway-event-filter>
      <gateway-transport-filter>
   <gateway-receiver>
      <gateway-transport-filter>
   <gateway-conflict-resolver>
      <async-event-queue>
      <async-event-listener>
         <class-name>
         <parameter>
            <string>
            <declarable>
   <cache-server>
      <group>
      <client-subscription>
      <custom-load-probe>
          <class-name>
          <parameter>
             <string>
             <declarable>
   <pool>
      <locator>
      <server>
   <disk-store>
      <disk-dirs>
         <disk-dir>
   <pdx>
      <pdx-serializer>
         <class-name>
         <parameter>
            <string>
            <declarable>
   <region-attributes>
      <key-constraint>
      <value-constraint>
```

```
       <region-time-to-live>
          <expiration-attributes>
             <custom-expiry>
                <class-name>
                <parameter>
                   <string>
                   <declarable>
       <region-idle-time>
          <expiration-attributes>
             <custom-expiry>
                <class-name>
                <parameter>
                   <string>
                   <declarable>
       <entry-time-to-live>
          <expiration-attributes>
             <custom-expiry>
                <class-name>
                <parameter>
                   <string>
                   <declarable>
       <entry-idle-time>
          <expiration-attributes>
             <custom-expiry>
                <class-name>
                <parameter>
                   <string>
                   <declarable>
       <partition-attributes>
          <partition-resolver>
             <class-name>
                <parameter>
                   <string>
                   <declarable>
          <partition-listener>
             <class-name>
                <parameter>
                   <string>
                   <declarable>
          <fixed-partition=attributes>
       <membership-attributes>
          <required-role>
       <subscription-attributes>
          <cache-loader>
             <class-name>
             <parameter>
                <string>
                <declarable>
          <cache-writer>
             <class-name>
             <parameter>
                <string>
                <declarable>
          <cache-listener>
             <class-name>
             <parameter>
                <string>
                <declarable>
       <compressor>
```

```
         <class-name>
         <parameter>
            <string>
            <declarable>
      <eviction-attributes>
         <lru-entry-count>
         <lru-heap-percentage>
            <class-name>
               <parameter>
                  <string>
                  <declarable>
         <lru-memory-size>
            <class-name>
               <parameter>
                  <string>
                  <declarable>
   <jndi-bindings>
      <jndi-binding>
         <config-property>
            <config-property-name>
            <config-property-type>
            <config-property-value>
   <region>
      <region-attributes>
      <index>
      <lucene:index>
         <field>
      <entry>
         <key>
            <string>
            <declarable>
         <value>
            <string>
            <declarable>
   <vm-root-region>
   <function-service>
      <function>
         <class-name>
         <parameter>
            <string>
            <declarable>
   <resource-manager>
   <serialization-registration>
      <serializer>
         <class-name>
            <instantiator>
         <class-name>
            <backup>
   <initializer><initializer>
      <class-name>
      <parameter>
         <string>
         <declarable>
</cache>
```

<cache> Element Reference

This section documents the `cache.xml` sub-elements used for VMware GemFire server configuration. All elements are sub-elements of the `<cache>` element.

For VMware GemFire client configuration, see <client-cache> Element Reference.

API:`org.apache.geode.cache.CacheFactory`

<cache> Attributes

Attribute	Description	Default
copy-on-read	Boolean indicating whether entry value retrieval methods return direct references to the entry value objects in the cache (false) or copies of the objects (true).	False
is-server	Boolean indicating whether this member is a cache server.	False
lock-timeout	The timeout, in seconds, for implicit object lock requests. This setting affects automatic locking only, and does not apply to manual locking. If a lock request does not return before the specified timeout period, it is cancelled and returns with a failure.	60
lock-lease	The timeout, in seconds, for implicit and explicit object lock leases. This affects both automatic locking and manual locking. Once a lock is obtained, it can remain in force for the lock lease time period before being automatically cleared by the system.	120
message-sync-interval	Used for client subscription queue synchronization when this member acts as a server to clients and server redundancy is used. Sets the frequency (in seconds) at which the primary server sends messages to its secondary servers to remove queued events that have already been processed by the clients.	1
search-timeout	How many seconds a `netSearch` operation can wait for data before timing out. You may want to change this based on your knowledge of the network load or other factors.	300

Example:

```
<cache>
  <cache-server port="40404" />
  <region name="root">
    <region-attributes refid="REPLICATE"/>
    <region name="cs_region" refid="REPLICATE">
      <region-attributes>
        <cache-loader>
          <class-name>cacheRunner.StringLoader</class-name>
        </cache-loader>
          <cache-listener>
          <class-name>cacheRunner.LoggingCacheListener</class-name>
        </cache-listener>
      </region-attributes>
    </region>
  </region>
</cache>
```

<cache-transaction-manager>

Specifies a transaction listener.

API: `CacheTransactionManager`

Example:

```
<cache search-timeout="60">
   <cache-transaction-manager>
     <transaction-listener>
       <class-name>com.company.data.MyTransactionListener</class-name>
       <parameter name="URL">
         <string>jdbc:cloudscape:rmi:MyData</string>
       </parameter>
     </transaction-listener>
     <transaction-listener>... </transaction-listener>
     <transaction-writer>
       <class-name>com.company.data.MyTransactionWriter</class-name>
       <parameter name="URL">
         <string>jdbc:cloudscape:rmi:MyData</string>
       </parameter>
       <parameter>
     </transaction-writer>
   </cache-transaction-manager> .. .
</cache>
```

<transaction-listener>

When a transaction ends, its thread calls the TransactionListener to perform the appropriate follow-up for successful commits, failed commits, or voluntary rollbacks.

Specify the Java class and its initialization parameters with the `<class-name>` and `<parameter>` sub-elements. See <class-name> and <parameter>.

<transaction-writer>

When you commit a transaction, a TransactionWriter can perform additional tasks, including cancelling the transaction.

Specify the Java class and its initialization parameters with the `<class-name>` and `<parameter>` sub-elements. See <class-name> and <parameter>.

<dynamic-region-factory>

The `<dynamic-region-factory>` element configures a dynamic region factory for this cache. You can use this element to dynamically create regions in your application code. Use the `createDynamicRegion()` method of the `org.apache.geode.cache.DynamicRegionFactory` class in your Java code to dynamically create regions.

Note: You cannot use this element to dynamically create *partitioned* regions.

Note: Use of the `DynamicRegionFactory` class and the `<dynamic-region-factory>` element are deprecated in favor of the `org.apache.geode.cache.execute.FunctionService` class and the `<function-service>` element.

We recommend that you use functions to dynamically create regions. See Creating Regions Dynamically.

The optional `<disk-dir>` sub-element specifies the directory to store the persistent files that are used for dynamic region bookkeeping. It defaults to the current directory.

Set the `pool-name` attribute to set the name of the connection pool used by client applications in a client/server cache configuration. Do not specify the `pool-name` attribute in servers or peers.

API: `org.apache.geode.cache.DynamicRegionFactory`

<dynamic-region-factory> Attributes

Attribute	Description	Default
disable-persist-backup	When set to false, the factory is persisted on disk.	false
disable-register-interest	When set to false, client regions created by the factory register interest in all keys in a corresponding server cache region.	false
pool-name	The name of a connection pool used by the client factory to communicate with the server-side factory.	none

Example:

```
<dynamic-region-factory
     pool-name=myPool>
     <disk-dir>/home/gemfire/myDiskdir</disk-dir>
</dynamic-region-factory>
```

<disk-dir>

Specifies a region or disk store’s disk directory.

<disk-dir> Attributes

Attribute	Description	Default
dir-size	Maximum amount of space to use for the disk store, in megabytes.	214748364 (2 petabytes)

Example:

```
<disk-dir
    dir-size="20480">/host3/users/gf/memberA_DStore</disk-dir>
```

<gateway-sender>

Configures a gateway sender to distribute region events to another VMware GemFire site. See Configuring a Multi-site (WAN) System.

API: `GatewaySender`

<gateway-sender> Attributes

Attribute	Description	Default
parallel	Value of "true" or "false" that specifies the type of gateway sender that VMware GemFire creates.	false
dispatcher-threads	Number of dispatcher threads that are used to process region events from a gateway sender queue or asynchronous event queue.	5
order-policy	When the `dispatcher-threads` attribute is greater than 1, `order-policy` configures the way in which multiple dispatcher threads process region events from a serial gateway queue or serial asynchronous event queue. This attribute can have one of the following values: • **key**. When distributing region events from the local queue, multiple dispatcher threads preserve the order of key updates. • **thread**. When distributing region events from the local queue, multiple dispatcher threads preserve the order in which a given thread added region events to the queue. • **partition**. This option is valid for parallel event queues. When distributing region events from the local queue, multiple dispatcher threads preserve the order in which region events were added to the local queue. For a partitioned region, this means that all region events delivered to a specific partition are delivered in the same order to the remote VMware GemFire site. For a distributed region, this means that all key updates delivered to the local queue are distributed to the remote site in the same order. You cannot configure the `order-policy` for a parallel event queue, because parallel queues cannot preserve event ordering for regions. Only the ordering of events for a given partition (or in a given queue of a distributed region) can be preserved.	key
id	Unique identifier for the gateway sender, usually an identifier associated with a physical location. This attribute is required.	null
remote-distributed-system-id	Integer that uniquely identifies the remote VMware GemFire cluster to which this gateway sender will send region events. This value corresponds to the `distributed-system-id` property specified in locators for the remote cluster. This attribute is required.	null

Attribute	Description	Default
manual-start	**Deprecated.** Boolean value that specifies whether you need to manually start the gateway sender. If you supply a null value, the default value of false is used, and the gateway sender attempts to start automatically. *A manual start is likely to cause data loss, so manual start should never be used in a production system.*	false
socket-buffer-size	Size of the socket buffer that sends messages to remote sites. This size should match the size of the `socket-buffer-size` attribute of remote gateway receivers that process region events.	32768
socket-read-timeout	Amount of time in milliseconds that the gateway sender will wait to receive an acknowledgment from a remote site. By default this is set to 0, which means there is no timeout. If you do set this timeout, you must set it to a minimum of 30000 (milliseconds). Setting it to a lower number will generate an error message and reset the value to the default of 0.	0
enable-batch-conflation	Boolean value that determines whether VMware GemFire should conflate messages.	false
batch-size	Maximum number of messages that a batch can contain.	100
batch-time-interval	Maximum number of milliseconds that can elapse between sending batches.	1000
enable-persistence	Boolean value that determines whether VMware GemFire persists the gateway queue.	false
disk-store-name	Named disk store to use for storing the queue overflow, or for persisting the queue. If you specify a value, the named disk store must exist. If you specify a null value, VMware GemFire uses the default disk store for overflow and queue persistence.	
disk-synchronous	For regions that write to disk, boolean that specifies whether disk writes are done synchronously for the region.	true
maximum-queue-memory	Maximum amount of memory in megabytes that the queue can consume before overflowing to disk.	100 MB
alert-threshold	Maximum number of milliseconds that a region event can remain in the gateway sender queue before VMware GemFire logs an alert.	0
group-transaction-events	Boolean value to ensure that all the events of a transaction are sent in the same batch, i.e., they are never spread across different batches. Only allowed to be set on gateway senders with the `parallel` attribute set to false and `dispatcher-threads` attribute equal to 1, or on gateway senders with the `parallel` attribute set to true. **Note:** In order to work for a transaction, the regions to which the transaction events belong must be replicated by the same set of senders with this flag enabled.	false

Example:

```
<cache>

  <gateway-sender
    id="remoteA"
    parallel="true"
    remote-distributed-system-id="1">
    <gateway-event-filter>
      <class-name>org.apache.geode.util.SampleEventFilter</class-name>
      <parameter
       name="param1">
        <string>"value1"</string>
      </parameter>
    </gateway-event-filter>
    <gateway-transport-filter>
      <class-name>org.apache.geode.util.SampleTransportFilter</class-name>
      <parameter
       name="param1">
        <string>"value1"</string>
      </parameter>
    </gateway-transport-filter>
  </gateway-sender>
  ...
</cache>
```

<gateway-event-filter>

A GatewayEventFilter implementation determines whether a region event is placed in a gateway sender queue and/or whether an event in a gateway queue is distributed to a remote site. You can optionally add one or more GatewayEventFilter implementations to a gateway sender, either in the cache.xml configuration file or using the Java API.

Specify the Java class and its initialization parameters with the `<class-name>` and `<parameter>` sub-elements. See <class-name> and <parameter>.

Example:

```
<gateway-event-filter>
      <class-name>org.apache.geode.util.SampleEventFilter</class-name>
      <parameter name="param1">
        <string>"value1"</string>
      </parameter>
</gateway-event-filter>
```

<gateway-event-substitution-filter>

A GatewayEventSubstitutionFilter provides a way to specify a substitute value to be stored in the GatewayQueueEvent and enqueued in the RegionQueue. You can optionally add one GatewayEventSubstitutionFilter implementation to a gateway sender.

Specify the Java class and its initialization parameters with the `<class-name>` and `<parameter>` sub-elements. See <class-name> and <parameter>.

Example:

```
<gateway-event-substitution-filter>
      <class-name>org.apache.geode.util.SampleEventSubstitutionFilter</class-name>
      <parameter name="param1">
        <string>"value1"</string>
      </parameter>
</gateway-event-substitution-filter>
```

<gateway-transport-filter>

Use a GatewayTransportFilter implementation to process the TCP stream that sends a batch of events that is distributed from one VMware GemFire cluster to another over a WAN. A GatewayTransportFilter is typically used to perform encryption or compression on the data that distributed. You install the same GatewayTransportFilter implementation on both a gateway sender and gateway receiver.

Specify the Java class and its initialization parameters with the `<class-name>` and `<parameter>` sub-elements. See <class-name> and <parameter>.

Example:

```
<gateway-transport-filter>
      <class-name>org.apache.geode.util.SampleTransportFilter</class-name>
      <parameter
       name="param1">
        <string>"value1"</string>
      </parameter>
</gateway-transport-filter>
```

<gateway-receiver>

Configures a gateway receiver to receive and apply region events that were distributed from another VMware GemFire site. You can only specify one gateway receiver on a member. See Configuring a Multi-site (WAN) System.

API: `GatewayReceiverFactory`, `GatewayTransportFilter`

<gateway-receiver> Attributes

Attribute	Description	Default
start-port	Starting port number to use when specifying the range of possible port numbers this gateway receiver will use to connects to gateway senders in other sites. VMware GemFire chooses an unused port number in the specified port number range to start the receiver. If no port numbers in the range are available, an exception is thrown. The `STARTPORT` value is inclusive while the `ENDPORT` value is exclusive. For example, if you specify `STARTPORT="50510"` and `ENDPOINT="50520"`, VMware GemFire chooses a port value from 50510 to 50519.	5000

Attribute	Description	Default
end-port	Defines the upper bound port number to use when specifying the range of possible port numbers this gateway receiver will use to for connections from gateway senders in other sites. VMware GemFire chooses an unused port number in the specified port number range to start the receiver. If no port numbers in the range are available, an exception is thrown. The `ENDPORT` value is exclusive while the `STARTPORT` value is inclusive. For example, if you specify `STARTPORT="50510"` and `ENDPOINT="50520"`, VMware GemFire chooses a port value from 50510 to 50519.	5500
bind-address	Network address for connections from gateway senders in other sites. Specify the address as a literal string value.	
hostname-for-senders	Attribute where you can specify an IP address or hostname for gateway sender connections. If you configure hostname-for-senders, locators will use the provided hostname or IP address when instructing gateway senders on how to connect to gateway receivers. If you provide "" or null as the value, by default the gateway receiver's bind-address will be sent to clients.	
manual-start	When set to false, the gateway receiver will automatically start when the receiver is created. If set to true, you must manually start the receiver.	true
maximum-time-between-pings	Integer value that specifies the time interval (in milliseconds) to use between pings to connected WAN sites. This value determines the maximum amount of time that can elapse before a remote WAN site is considered offline.	60000
socket-buffer-size	An integer value that sets the buffer size (in bytes) of the socket connection for this gateway receiver. This value should match the `socket-buffer-size` setting of gateway senders that connect to this receiver.	32768

Example:

```
<cache>
     <gateway-receiver start-port="1530" end-port="1551">
    <gateway-transport-filter>>
      <class-name>org.apache.geode.util.SampleTransportFilter</class-name>
      <parameter
       name="param1">
        <string>"value1"</string>
      </parameter>
    </gateway-transport-filter>
  </gateway-receiver>
</cache>
```

<gateway-transport-filter>

Use a GatewayTransportFilter implementation to process the TCP stream that sends a batch of events that is distributed from one VMware GemFire cluster to another over a WAN. A GatewayTransportFilter is typically used to perform encryption or compression on the data that distributed. You install the same GatewayTransportFilter implementation on both a gateway sender and gateway receiver.

Specify the Java class and its initialization parameters with the `<class-name>` and `<parameter>` sub-elements. See <class-name> and <parameter>.

Example:

```
<gateway-transport-filter>
      <class-name>org.apache.geode.util.SampleTransportFilter</class-name>
      <parameter
       name="param1">
        <string>"value1"</string>
      </parameter>
</gateway-transport-filter>
```

<gateway-conflict-resolver>

An event-handler plug-in that is called in order to determine whether a potentially conflicting WAN update should be applied to the local cache. A GatewayConflictResolver is invoked if the current value in a cache entry was established by a different cluster (with a different distributed-system-id) than an event that is attempting to modify the entry. It is not invoked if the event has the same distributed system ID as the event that last changed the entry. See Resolving Conflicting Events

Specify the Java class for the gateway conflict resolver plug-in and its initialization parameters with the `<class-name>` and `<parameter>` sub-elements. See <class-name> and <parameter>.

API: `org.apache.geode.cache.util.GatewayConflictResolver`

Example:

```
<gateway-conflict-resolver>
    <class-name>
     myPackage.MyConflictResolver
    </class-name>
  </gateway-conflict-resolver>
```

<async-event-queue>

Configures a queue for sending region events to an AsyncEventListener implementation (for example, for write-behind event handling).

API: `org.apache.geode.cache.asyncqueue.AsyncEventQueue`

<async-event-queue> Attributes

Attribute	Description	Default
id	Unique identifier for the queue. This attribute is required.	null
parallel	Value of "true" or "false" that specifies the type of queue that VMware GemFire creates.	false

Attribute	Description	Default
batch-size	Maximum number of messages that a batch can contain.	100
batch-time-interval	Maximum number of milliseconds that can elapse between sending batches.	5
enable-batch-conflation	Boolean value that determines whether VMware GemFire should conflate messages.	false
disk-store-name	Named disk store to use for storing queue overflow, or for persisting the queue. If you specify a value, the named disk store must exist. If you specify a null value, VMware GemFire uses the default disk store for overflow and queue persistence.	null specifies the default disk store
disk-synchronous	For regions that write to disk, boolean that specifies whether disk writes are done synchronously for the region.	true
dispatcher-threads	Number of dispatcher threads that are used to process region events from the queue.	5
forward-expiration-destroy	When true, forwards expiration destroy operations to AsyncEventListener.	false
maximum-queue-memory	Maximum amount of memory in megabytes that the queue can consume before overflowing to disk.	100 mb
order-policy	When the `dispatcher-threads` attribute is greater than 1, `order-policy` configures the way in which multiple dispatcher threads process region events from the queue. This attribute can have one of the following values: • **key**. When distributing region events from the local queue, multiple dispatcher threads preserve the order of key updates. • **thread**. When distributing region events from the local queue, multiple dispatcher threads preserve the order in which a given thread added region events to the queue. • **partition**. This option is valid for parallel event queues. When distributing region events from the local queue, multiple dispatcher threads preserve the order in which region events were added to the local queue. For a partitioned region, this means that all region events delivered to a specific partition are delivered in the same order to the remote VMware GemFire site. For a distributed region, this means that all key updates delivered to the local queue are distributed to the remote site in the same order. You cannot configure the `order-policy` for a parallel event queue, because parallel queues cannot preserve event ordering for regions. Only the ordering of events for a given partition (or in a given queue of a distributed region) can be preserved.	key

Attribute	Description	Default
pause-event-processing	When true, event dispatching from the queue to the listener(s) will be paused when the AsyncEventQueue is started.	false
persistent	Boolean value that determines whether VMware GemFire persists this queue.	False

Example:

```
<cache>
  <async-event-queue
    id="sampleQueue"
    persistent="true"
    disk-store-name="exampleStore"
    parallel="false">
    <async-event-listener>
      <class-name>MyAsyncEventListener</class-name>
      <parameter name="url">
        <string>jdbc:db2:SAMPLE</string>
      </parameter>
      <parameter name="username">
        <string>gfeadmin</string>
      </parameter>
      <parameter name="password">
        <string>admin1</string>
      </parameter>
    </async-event-listener>
  </async-event-queue>
  ...
</cache>
```

<async-event-listener>

An AsyncEventListener receives callbacks for events that change region data. You can use an AsyncEventListener implementation as a write-behind cache event handler to synchronize region updates with a database.

Specify the Java class and its initialization parameters with the `<class-name>` and `<parameter>` sub-elements. See <class-name> and <parameter>.

API: `org.apache.geode.cache.asyncqueue.AsyncEventListener`

Example:

```
...
  <async-event-listener>
    <class-name>MyAsyncEventListener</class-name>
    <parameter name="url">
      <string>jdbc:db2:SAMPLE</string>
    </parameter>
    <parameter name="username">
      <string>gfeadmin</string>
    </parameter>
    <parameter name="password">
      <string>admin1</string>
    </parameter>
  </async-event-listener>
...
```

<cache-server>

Configures the cache to serve region data to clients in a client/server caching system. This element indicates the port the server listens on for client communication.

The `cacheserver` process uses only `cache.xml` configuration. For application servers, you can set the same configuration properties using the `org.apache.geode.cache.server.CacheServer` and `org.apache.geode.cache.Cache` interfaces. For detailed information, see the online Java API documentation.

API: `org.apache.geode.cache.server.CacheServer`

<cache-server> Attributes

Attribute	Description	Default Value
bind-address	Hostname or IP address that the server is to listen on for client connections. If null, the server listens on the machine's default address.	null
hostname-for-clients	Hostname or IP address to pass to the client as the location where the server is listening. When the server connects to the locator it tells the locator the host and port where it is listening for client connections. If the host the server uses by default is one that the client can't translate into an IP address, the client will have no route to the server's host and won't be able to find the server. For this situation, you must supply the server's alternate hostname for the locator to pass to the client. If null, the server's `bind-address` setting is used.	null
load-poll-interval	Frequency, in milliseconds, to poll the load probe for load information on the server.	5000 (5 seconds)
max-connections	Maximum number of client connections for the server. When the maximum is reached, the server refuses additional client connections. **Note:** Set this at least as high as max-threads.	800
max-threads	Maximum number of threads allowed in this server to service client connections. When the limit is reached, server threads begin servicing multiple connections. A zero setting causes the server to use a thread for every client connection. **Note:** Set this no higher than max-connections.	0
maximum-message-count	Maximum number of messages allowed in a subscription queue. When the queue reaches this limit, messages block. **Note:** Used only if `client-subscription` is not configured.	230000

Attribute	Description	Default Value
maximum-time-between-pings	Maximum time, in milliseconds, the server allows to pass between messages or pings indicating a client is healthy. **Note:** A setting of 0 or a negative number turns off client health monitoring. Be careful not to do this accidentally.	60000 (1 minute)
message-time-to-live	Setting used for highly available subscription queues. The expiration time, in seconds, for non-durable messages in the secondary server's client subscription queue. The system removes non-durable messages that have been in the queue beyond this time. If set to 0 (zero), the messages are never removed. **Note:** Set this high enough to avoid removing messages that are still valid, to aovid losing messages during server failover.	180 (3 minutes)
port	Port that the server listens on for client communication.	40404
socket-buffer-size	Size for socket buffers used for server-to-client communication.	32768
tcp-no-delay	When set to true, enables TCP_NODELAY for VMware GemFire server connections to clients.	false

Example:

```
<cache>
  <cache-server
    port="40404"
  />
  ...
</cache>
```

<client-subscription>

Overflow specification for client subscription queues. Sets a capacity limit on the in-memory queue and specifies where to overflow when capacity is reached. By default no overflow is used. Specified in three parts:

Default: no overflow

API: `org.apache.geode.cache.server.ClientSubscriptionConfig`

<client-subscription> Attributes

Attribute	Description	Default
eviction-policy	How the capacity is calculated. The options are mem for memory use, entry for message count, and null for no overflow.	null
capacity	Used if eviction-policy is not null. Specified in megabytes for mem and as a positive integer for entry.	1
disk-store-name	Used if eviction-policy is not null. Default: default disk store. If specified, the disk-store-name must specify a disk store that is already defined in the cache.	default disk store

Example:

```
<cache>
  <cache-server port=4444>
    <client-subscription eviction-policy="entry | mem" capacity=35 overflow-directory
="OverflowDir"></client-subscription>
    ...
  </cache-server>
</cache>
```

<custom-load-probe>

Application plug-in used to provide current and predicted server load information to the server locators.

Specify the Java class and its initialization parameters with the `<class-name>` and `<parameter>` sub-elements. See <class-name> and <parameter>.

Default: If this is not defined, the default VMware GemFire load probe is used.

API: `org.apache.geode.cache.server.setLoadProbe`

Example:

```
<custom-load-probe>
  <class-name>
    myPackage.MyLoadProbe
  </class-name>
</custom-load-probe>
```

<pool>

Use for client caches. Defines a client's server pool used to communicate with servers running in a different cluster.

API: `org.apache.geode.cache.client.PoolFactory`

<pool> Attributes

Attribute	Description	Default
free-connection-timeout	Amount of time a thread will wait to get a pool connection before timing out with an exception. This timeout keeps threads from waiting indefinitely when the pool's `max-connections` has been reached and all connections in the pool are in use by other threads.	10000
idle-timeout	Maximum time, in milliseconds, a pool connection can stay open without being used when there are more than `min-connections` in the pool. Pings over the connection do not count as connection use. If set to -1, there is no idle timeout.	5000
load-conditioning-interval	Amount of time, in milliseconds, a pool connection can remain open before being eligible for silent replacement to a less-loaded server.	300000 (5 minutes)

Attribute	Description	Default
max-connections	Maximum number of pool connections the pool can create. If the maximum connections are in use, an operation requiring a client-to-server connection blocks until a connection becomes available or the `free-connection-timeout` is reached. If set to -1, there is no maximum. The setting must indicate a cap greater than `min-connections`. **Note:** If you need to use this to cap your pool connections, you should disable the pool attribute `pr-single-hop-enabled`. Leaving single hop enabled can increase thrashing and lower performance.	-1
min-connections	Minimum number of pool connections to keep available at all times. Used to establish the initial connection pool. If set to 0 (zero), no connection is created until an operation requires it. This number is the starting point, with more connections added later as needed, up to the `max-connection` setting. The setting must be an integer greater than or equal to 0.	1
multiuser-authentication	Used for installations with security where you want to accommodate multiple users within a single client. If set to true, the pool provides authorization for multiple user instances in the same client application, and each user accesses the cache through its own `RegionService` instance. If false, the client either uses no authorization or just provides credentials for the single client process.	false
name	Name of this pool. Used in the client region pool-name to assign this pool to a region in the client cache. **Note:** This is a required property with no default setting.	none
ping-interval	How often to communicate with the server to show the client is alive, set in milliseconds. Pings are only sent when the ping-interval elapses between normal client messages. **Note:** Set this lower than the server's `maximum-time-between-pings`.	10000
pr-single-hop-enabled	Setting used to improve access to partitioned region data in the servers. Indicates whether to use metadata about the partitioned region data storage locations to decide where to send some data requests. This allows a client to send a data operation directly to the server hosting the key. Without this, the client contacts any available server and that server contacts the data store. This is used only for operations that can be carried out on a server-by-server basis, like put, get, and destroy.	true
read-timeout	Maximum time, in milliseconds, for the client to wait for a response from a server.	10000

Attribute	Description	Default
retry-attempts	Number of times to retry a client request before giving up. If one server fails, the pool moves to the next, and so on until it is successful or it hits this limit. If the available servers are fewer than this setting, the pool will retry servers that have already failed until it reaches the limit. If this is set to -1, the pool tries every available server once.	-1
server-group	Logical named server group to use from the pool. A null value uses the global server group to which all servers belong. **Note:** This is only used when the `locator` list is defined.	null
socket-buffer-size	Size for socket buffers from the client to the server. Default: 32768.	32768
statistic-interval	Interval, in milliseconds, at which to send client statistics to the server. If set to -1, statistics are not sent.	-1
subscription-ack-interval	Time, in milliseconds, between messages to the primary server to acknowledge event receipt. **Note:** Used only when `subscription-redundancy` is not '0' (zero).	100
subscription-enabled	Boolean indicating whether the server should connect back to the client and automatically sends server-side cache update information. Any bind address information for the client is automatically passed to the server for use in the callbacks.	false
subscription-message-tracking-timeout	Time-to-live, in milliseconds, for entries in the client's message tracking list.	900000 (15 minutes)
subscription-redundancy	Number of servers to use as backup to the primary for highly available subscription queue management. If set to 0, none are used. If set to -1, all available servers are used.	0

Example:

```
<pool
  name="publisher"
  subscription-enabled="true">
  <locator
    host="myLocatorAddress1"
    port="12345"/>
  <locator
    host="myLocatorAddress2"
    port="45678"/>
</pool>
```

<locator>

Addresses and ports of the locators to connect to. You can define multiple locators for the pool.

Note: Provide a locator list or `server` list, but not both.

API: `org.apache.geode.distributed.LocatorLauncher`

<locator> Attributes

Attribute	Description	Default
host	Hostname of the locator	
port	Port number of the locator	

Example:

```
<pool ...>
  <locator
    host="myLocatorHost"
    port="12345"/>
```

<server>

Addresses and ports of the servers to connect to.

Note: Provide a server list or `locator` list, but not both.

Default:

API: `org.apache.geode.distributed.ServerLauncher`

<server> Attributes

Attribute	Description	Default
host	Hostname of the server	
port	Port number of the server	

Example:

```
<pool ...>
  <server
    host="myServerHost"
    port="123456"/>
</pool>
```

<disk-store>

Defines a pool of one or more disk stores, which can be used by regions, and client subscription queues.

Default: The cache default disk store, named “DEFAULT”, is used when disk is used but no disk store is named.

API: `org.apache.geode.cache.DiskStore`

<disk-store> Attributes

Table 10. <disk-store> Attributes

Attribute	Description	Default

Attribute	Description	Default
name	The name of the Disk Store.	
auto-compact	Set to true to automatically compact the disk files.	
compaction-threshold	The threshold at which an oplog will become compactable. Until it reaches this threshold the oplog will not be compacted. The threshold is a percentage in the range 0 to 100.	
allow-force-compaction	Set to true to allow disk compaction to be forced on this disk store.	
max-oplog-size	The maximum size, in megabytes, of an oplog (operation log) file.	
time-interval	The number of milliseconds that can elapse before unwritten data is written to disk.	
write-buffer-size	The size of the write buffer that this disk store uses when writing data to disk. Larger values may increase performance but use more memory. The disk store allocates one direct memory buffer of this size.	
queue-size	Maximum number of operations that can be asynchronously queued to be written to disk.	
disk-usage-warning-percentage	Disk usage above this threshold generates a warning message. For example, if the threshold is set to 90%, then on a 1 TB drive falling under 100 GB of free disk space generates the warning. Set to "0" (zero) to disable.	90
disk-usage-critical-percentage	Disk usage above this threshold generates an error message and shuts down the member's cache. For example, if the threshold is set to 99%, then falling under 10 GB of free disk space on a 1 TB drive generates the error and shuts down the cache. Set to "0" (zero) to disable.	99

Example:

```
<disk-store
  name="DEFAULT"
  allow-force-compaction="true">
  <disk-dirs>
    <disk-dir>/export/thor/customerData</disk-dir>
    <disk-dir>/export/odin/customerData</disk-dir>
    <disk-dir>/export/embla/customerData</disk-dir>
  </disk-dirs>
</disk-store>
```

<disk-dirs>

An element of a disk store that defines a set of `<disk-dir>` elements.

<disk-dir>

Specifies a region or disk store's disk directory.

<disk-dir> Attributes

Attribute	Description	Default
dir-size	Maximum amount of space to use for the disk store, in megabytes.	214748364 (2 petabytes)

Example:

```
<disk-dir
  dir-size="20480">/host3/users/gf/memberA_DStore</disk-dir>
```

<pdx>

Specifies the configuration for the Portable Data eXchange (PDX) method of serialization.

API: `org.apache.geode.cache.CacheFactory.setPdxReadSerialized`, `setPdxDiskStore`, `setPdxPersistent`, `setPdxIgnoreUnreadFields` and `org.apache.geode.cache.ClientCacheFactory.setPdxReadSerialized`, `setPdxDiskStore`, `setPdxPersistent`, `setPdxIgnoreUnreadFields`

<pdx> Attributes

Attribute	Description	Default
read-serialized	Set it to true if you want PDX deserialization to produce a PdxInstance instead of an instance of the domain class.	
ignore-unread-fields	Set it to true if you do not want unread PDX fields to be preserved during deserialization. You can use this option to save memory. Set to true only in members that are only reading data from the cache.	
persistent	Set to true if you are using persistent regions. This causes the PDX type information to be written to disk.	
disk-store-name	If using persistence, this attribute allows you to configure the disk store that the PDX type data will be stored in. By default, the default disk store is used.	

Example:

```
<cache>
  <pdx persistent="true" disk-store-name="myDiskStore">
    <pdx-serializer>
      <class-name>
       org.apache.geode.pdx.ReflectionBasedAutoSerializer
      </class-name>
    <parameter name="classes">
      <string>com.company.domain.DomainObject</string>>
    </parameter>
  </pdx-serializer>
 </pdx>
```

```
  ...
</cache>
```

<pdx-serializer>

Allows you to configure the PdxSerializer for this VMware GemFire member.

Specify the Java class and its initialization parameters with the `<class-name>` and `<parameter>` sub-elements. See <class-name> and <parameter>.

Default:

API: `org.apache.geode.cache.CacheFactory.setPdxSerializer`

Example:

```
<cache>
  <pdx>
    <pdx-serializer>
     <class-name>com.company.ExamplePdxSerializer</class-name>
    </pdx-serializer>
  </pdx>
  ...
</cache>
```

<region-attributes>

Specifies a region attributes template that can be named (by `id`) and referenced (by `refid`) later in the `cache.xml` and through the API.

API: `org.apache.geode.cache.RegionFactory` or `org.apache.geode.cache.ClientRegionFactory`

<region-attributes> Attributes

Attribute	Description	Default

concurrency-level	Gives an estimate of the maximum number of application threads that will concurrently access a region entry at one time. This attribute does not apply to partitioned regions. This attribute helps VMware GemFire optimize the use of system resources and reduce thread contention. This sets an initial parameter on the underlying `java.util.ConcurrentHashMap` used for storing region entries.
	Note: Before you modify this, read the concurrency level description, then see the Java API documentation for `java.util.ConcurrentHashMap`.
	API: `setConcurrencyLevel`
	Example:
	`<region-attributes`
	`  concurrency-level="10">`
	`</region-attributes>`
		16 (threads)

Attribute	Description	Default

data-policy	Specifies how the local cache handles data for a region. This setting controls behavior such as local data storage and region initialization.
	
	Note: Configure the most common options using the region shortcuts, `RegionShortcut` and `ClientRegionShortcut`. The default `data-policy` of `normal` specifies local cache storage. The empty policy specifies no local storage. In the region shortcuts, empty corresponds to the settings with the string `PROXY`. You can use an empty region for event delivery to and from the local cache without the memory overhead of data storage.
	
	You can specify the following data policies:
	
	empty: No data storage in the local cache. The region always appears empty. Use this for event delivery to and from the local cache without the memory overhead of data storage - zero-footprint producers that only distribute data to others and zero-footprint consumers that only see events. To receive events with this, set the region's `subscription-attributes` `interest-policy` to all.
	
	normal: Data used locally (accessed with `get`, stored with `put`, etc.) is stored in the local cache. This policy allows the contents in the cache to differ from other caches.
	
	partition: Data is partitioned across local and remote caches using the automatic data distribution behavior of partitioned regions. Additional configuration is done in the `partition-attributes`.
	
	replicate: The region is initialized with the data from other caches. After initialization, all events for the distributed region are automatically copied into the local region, maintaining a replica of the entire distributed region in the local cache. Operations that would cause the contents to differ with other caches are not allowed. This is compatible with local `scope`, behaving the same as for normal.
	
	persistent-partition: Behaves the same as `partition` and also persists data to disk.
	
	persistent-replicate: Behaves the same as `replicate` and also persists data to disk.
	
	preloaded: Initializes like a replicated region, then, once initialized, behaves like a normal region.
	
	API: `setDataPolicy`
	
	Example:
	
	`<region-attributes`
	`  data-policy="replicate">`
		normal

Attribute	Description	Default

	`</region-attributes>`
	
	This is similar to using a region shortcut with`refid`, however when you use the REPLICATE region shortcut, it automatically sets the region's scope to `distributed-ack`.
	
	`<region-attributes`
	`  refid="REPLICATE">`
	`</region-attributes>`
	
	If you use `data-policy`, you must set the scope explicitly.	

enable-async-conflation	For TCP/IP distributions between peers, specifies whether to allow aggregation of asynchronous messages sent by the producer member for the region. This is a special-purpose Boolean attribute that applies only when asynchronous queues are used for slow consumers. A false value disables conflation so that all asynchronous messages are sent individually. This special-purpose attribute gives you extra control over peer-to-peer communication between distributed regions using TCP/IP. This attribute does not apply to client/server communication or to communication using the UDP unicast or IP multicast protocols.
	
	Note: To use this attribute, the `multicast-enabled` region attribute `disable-tcp` in `gemfire.properties` must be false (the default for both). In addition, asynchronous queues must be enabled for slow consumers, specified with the `async`* gemfire properties.
	
	API: `setEnableAsyncConflation`
	
	Example:
	
	`<region-attributes`
	`  enable-async-conflation="false">`
	`</region-attributes>`
		true

enable-subscription-conflation	Boolean for server regions that specifies whether the server can conflate its messages to the client. A true value enables conflation.
	
	Note:
	
	The client can override this setting with the `conflate-events` property in its `gemfire.properties`.
	
	API: `setEnableSubscriptionConflation`
	
	Example:
	
	`<region-attributes`
	` enable-subscription-conflation="true">`
	` </region-attributes>`
		false

Attribute	Description	Default

gateway-sender-ids	Specifies one or more gateway sender IDs to use for distributing region events to remote VMware GemFire sites. Specify multiple IDs as a comma-separated list.
	
	API: `addGatewaySenderId`
	
	Example:
	
	`<region-attributes`
	`  gateway-sender-ids="nwsender,swsender">`
	`</region-attributes>`
		not set

async-event-queue-ids	Specifies one or more asynchronous event queues to use for distributing region events an `AsyncEventListener` implementation (for example, for write-behind cache event handling). Specify multiple IDs as a comma-separated list.
	
	API: `addAsyncEventQueueId`
	
	Example:
	
	`<region-attributes`
	`  async-event-queue-ids="customerqueue,ordersqueu`
	`e">`
	`</region-attributes>`
		not set

hub-id	If the `enable-gateway` attribute is set to true, a comma-separated list of gateway hub IDs that receive events from the region. Used only with GemFire version 6.x gateway configurations. For GemFire 7.0 configuration, see the `gateway-sender-id` attribute of the `<region-attributes>` element.	null

id	Stores the region attribute settings in the cache with this identifier. Once stored, the attributes can be retrieved using the region attribute `refid`.
	
	API: `setId`
	
	Example:
	
	`<region-attributes`
	`  id="persistent-replicated">`
	`</region-attributes>`
		not set

Attribute	Description	Default

ignore-jta	Boolean that determines whether operations on this region participate in active JTA transactions or ignore them and operate outside of the transactions. This is primarily used in cache loaders, writers, and listeners that need to perform non-transactional operations on a region, such as caching a result set.
	
	API: `setIgnoreJTA`
	
	Example:
	
	`<region-attributes`
	`  ignore-jta="true">`
	`</region-attributes>`
		false

index-update-type	Specifies whether region indexes are maintained synchronously with region modifications, or asynchronously in a background thread. In the `cache.xml` file, this is set as a value, asynchronous or synchronous, assigned to the `index-update-type` region attribute. Set this through the API by passing a boolean to the `setIndexMaintenanceSynchronous` method.
	
	API: `setIndexMaintenanceSynchronous`
	
	Example:
	
	`<region-attributes`
	`  index-update-type="asynchronous">`
	`</region-attributes>`
		synchronous updates

initial-capacity	Together with the `load-factor` region attribute, sets the initial parameters on the underlying `java.util.ConcurrentHashMap` used for storing region entries.
	
	API: `setInitialCapacity`
	
	Example:
	
	`<region-attributes`
	`  initial-capacity="20">`
	`</region-attributes>`
		16

Attribute	Description	Default

is-lock-grantor	Determines whether this member defines itself as the lock grantor for the region at region creation time. This only specifies whether the member becomes lock grantor at creation and does not reflect the current state of the member's lock grantor status. The member's lock grantor status may change if another member subsequently defines the region with `is-lock-grantor` set to true. This attribute is only relevant for regions with `global scope`, as only they allow locking. It affects implicit and explicit locking.
	
	API: `setLockGrantor`
	
	Example:
	
	`<region-attributes`
	`  is-lock-grantor="true">`
	`</region-attributes>`
		false

load-factor	Together with the initial-capacity region attribute, sets the initial parameters on the underlying `java.util.ConcurrentHashMap` used for storing region entries. This must be a floating point number between 0 and 1, inclusive.
	
	Note:
	
	Before you set this attribute, read the discussion of initial capacity and load factor, then see the Java API documentation for `java.util.ConcurrentHashMap`.
	
	API: `setLoadFactor`
	
	Example:
	
	`<region-attributes`
	`  load-factor="0.85">`
	`</region-attributes>`
		.75

mirror-type	Deprecated	

multicast-enabled	Boolean that specifies whether distributed operations on a region should use multicasting. To enable this, multicast must be enabled for the cluster with the `mcast-port` `gemfire.properties` setting.
	
	API: `setMulticastEnabled`
	
	Example:
	
	`<region-attributes`
	`  multicast-enabled="true">`
	`</region-attributes>`
		false

Attribute	Description	Default
pool-name	Identifies the region as a client region and specifies the server pool the region is to use. The named pool must be defined in the client cache before the region is created. If this is not set, the region does not connect to the servers as a client region. **API:** `setPoolName` **Examples:** This declaration creates the region as a client region with a server pool named DatabasePool. This pool-name specification is required, as there are multiple pools in the client cache:	not set

```
<client-cache>
    <pool name="DatabasePool"
   subscription-enabled="true">
        ...
    </pool>
    <pool >
   name="OtherPool"
   subscription-enabled="true">
        ...
    </pool>
    <region ...
        <region-attributes
          pool-name="DatabasePool">
        </region-attributes>
        ...
```

This declaration creates the region as a client region assigned the single pool that is defined for the client cache. Here the pool-name specification is implied to be the only pool that exists in the cache:

```
<client-cache>
    <pool
   name="publisher"
   subscription-enabled="true">
        ...
    </pool>
    <region
   name="myRegion"
   refid="CACHING_PROXY">
    </region>
</client-cache>
```

Attribute	Description	Default

disk-store-name	Assigns the region to the disk store with this name from the disk stores defined for the cache. Persist region data to disk by defining the region as persistent using the Shortcut Attribute Options or data-policy settings. Overflow data to disk by implementing LRU eviction-attributes with an action of overflow to disk. Each disk store defines the file system directories to use, how data is written to disk, and other disk storage maintenance properties. In addition, the `disk-synchronous` region attribute specifies whether writes are done synchronously or asynchronously.
	API: `setDiskStoreName`
	Example:
	`<region-attributes`
	`  disk-store-name="myStoreA" >`
	`</region-attributes>`
		null

disk-synchronous	For regions that write to disk, boolean that specifies whether disk writes are done synchronously for the region.
	API: `setDiskSynchronous`
	Example:
	`<region-attributes`
	`  disk-store-name="myStoreA"`
	`  disk-synchronous="true">`
	`</region-attributes>`
		true

refid	Retrieves region shortcuts and user-defined named region attributes for attributes initialization
	API: `setRefId`
	Example:
	`<region-attributes`
	`  refid="persistent-replicated">`
	`  <!-- Override any stored`
	`    attribute settings that you`
	`    need to ... -->`
	`</region-attributes>`
		not set

Attribute	Description	Default

scope	Definition: Determines how updates to region entries are distributed to the other caches in the cluster where the region and entry are defined. Scope also determines whether to allow remote invocation of some of the region's event handlers, and whether to use region entry versions to provide consistent updates across replicated regions.
	
	Note:
	
	You can configure the most common of these options with VMware GemFire region shortcuts in `RegionShortcut` and `ClientRegionShortcut`.
	
	Note:
	
	Server regions that are not partitioned must be replicated with `distributed-ack` or `global` scope. The region shortcuts that specify `REPLICATE` have `distributed-ack` scope.
	
	Set one of the following scope values:
	
	local: No distribution. The region is visible only to threads running inside the member.
	
	distributed-no-ack: Events are distributed to remote caches with no acknowledgement required.
	
	distributed-ack: Events are distributed to remote caches with receipt acknowledgement required. Region entry versions are used to provide consistent updates across members of the cluster.
	
	global: Events are distributed to remote caches with global locking to ensure distributed cache consistency.
	
	API: `setScope`
	
	Example:
	
	`<region-attributes`
	`  scope="distributed-ack">`
	`</region-attributes>`
		distributed-no-ack

Attribute	Description	Default

statistics-enabled	Boolean specifying whether to gather statistics on the region. Must be true to use expiration on the region. VMware GemFire provides a standard set of statistics for cached regions and region entries, which give you information for fine-tuning your cluster. Unlike other VMware GemFire statistics, statistics for local and distributed regions are not archived and cannot be charted. They are kept in instances of `org.apache.geode.cache.CacheStatistics` and made available through the region and its entries through the `Region.getStatistics` and `Region.Entry.getStatistics` methods.
	
	API: `setStatisticsEnabled`
	
	Example:
	
	`<region-attributes`
	`  statistics-enabled="true">`
	`</region-attributes>`
		false

cloning-enabled	Determines how `fromDelta` applies deltas to the local cache for delta propagation. When true, the updates are applied to a clone of the value and then the clone is saved to the cache. When false, the value is modified in place in the cache.
	
	API: `setCloningEnabled`
	
	Example:
	
	`<region-attributes`
	`  cloning-enabled="true">`
	`</region-attributes>`
		false

concurrency-checks-enabled	Determines whether members perform checks to provide consistent handling for concurrent or out-of-order updates to distributed regions. See Consistency for Region Updates.
	
	Note: Applications that use a `client-cache` may want to disable concurrency checking in order to see all events for a region. VMware GemFire server members can continue using concurrency checks for the region, but they will pass all events to the client cache. This configuration ensures that the client sees all events, but it does not prevent the client cache from becoming out-of-sync with the server cache.
	
	API: `setConcurrencyChecksEnabled`
	
	Example:
	
	`<region-attributes`
	` concurrency-checks-enabled="true">`
	`</region-attributes>`
		true

Attribute	Description	Default

off-heap	Specifies that the region uses off-heap memory to store entry values, including values for region entries and queue entries. The region will still use heap memory for everything else, such as entry keys and the ConcurrentHashMap.
	
	API: `setOffHeap`
	
	Example:
	
	`<region-attributes`
	`  off-heap="true">`
	`</region-attributes>`
		false

<key-constraint>

Defines the type of object to be allowed for the region entry keys. This must be a fully-qualified class name. The attribute ensures that the keys for the region entries are all of the same class. If key-constraint is not used, the region's keys can be of any class. This attribute, along with value-constraint, is useful for querying and indexing because it provides object type information to the query engine.

Note: Set the constraint in every cache where you create or update the region entries. For client/server installations, match constraints between client and server and between clusters. The constraint is only checked in the cache that does the entry `put` or `create` operation. To avoid deserializing the object, the constraint is not checked when the entry is distributed to other caches.

Default: not set

API: `org.apache.geode.cache.RegionFactory.setKeyConstraint`

Example:

```
<region-attributes>
  <key-constraint>
    java.lang.String
  </key-constraint>
</region-attributes>
```

<value-constraint>

Defines the type of object to be allowed for the region entry values. This must be a fully-qualified class name. If value constraint isn't used, the region's value can be of any class. This attribute, along with `key-constraint`, is useful for querying and indexing because it provides object type information to the query engine.

Note: Set the constraint in every cache where you create or update the region entries. For client/server installations, match constraints between client and server and between clusters. The constraint is only checked in the cache that does the entry `put` or `create` operation. To avoid deserializing the object, the constraint is not checked when the entry is distributed to other caches.

Default: not set

API: `org.apache.geode.cache.RegionFactory.setValueConstraint`

Example:

```
<region-attributes>
  <value-constraint>
    cacheRunner.Portfolio
  </value-constraint>
</region-attributes>
```

<region-time-to-live>

Expiration setting that specifies how long the region can remain in the cache without anyone accessing or updating it.

Default: not set - no expiration of this type

API: `org.apache.geode.cache.RegionFactory.setRegionTimeToLive`

Example:

```
<region-attributes
  statistics-enabled="true">
  <region-time-to-live>
    <expiration-attributes
      timeout="3600"
      action="local-destroy"/>
  </region-time-to-live>
</region-attributes>
```

<expiration-attributes>

Within the `entry-time-to-live` or `entry-idle-time` element, this element specifies the expiration rules for removing old region entries that you are not using. You can destroy or invalidate entries, either locally or across the cluster. Within the `region-time-to-live` or `region-idle-time` element, this element specifies the expiration rules for the entire region.

API: See APIs for `entry-time-to-live`, `entry-idle-time`, `region-time-to-live`, `region-idle-time`

<expiration-attributes> Attributes

Attribute	Description	Default
timeout	Number of seconds before a region or an entry expires. If timeout is not specified, it defaults to zero (which means no expiration).	0

Attribute	Description	Default
action	Action that should take place when a region or an entry expires. Select one of the following expiration actions: **local-destroy**: Removes the region or entry from the local cache, but does not distribute the removal operation to remote members. You cannot use this action on partitioned region entries. **destroy**: Removes the region or entry completely from the cache. Destroy actions are distributed according to the region's distribution settings. Use this option when the region or entry is no longer needed for any application in the cluster. **invalidate**: Default expiration action. Marks an entry or all entries in the region as invalid. Distributes the invalidation according to the region's scope. This is the proper choice when the region or the entry is no longer valid for any application in the cluster. **local-invalidate**: Marks an entry or all entries in the region as invalid but does not distribute the operation. You cannot use this action on partitioned region entries. Local region invalidation is only supported for regions that are not configured as replicated regions.	invalidate

Example:

```
<region-attributes
  statistics-enabled="true">
   <region-time-to-live>
      <expiration-attributes
        timeout="60"
        action="local-destroy"/>
   </region-time-to-live>
</region-attributes>
```

<custom-expiry>

Specifies the custom class that implements `org.apache.geode.cache.CustomExpiry`. You define this class in order to override the region-wide settings for specific entries. See Configure Data Expiration for an example.

Specify the Java class and its initialization parameters with the `<class-name>` and `<parameter>` sub-elements.

API: `org.apache.geode.cache.RegionFactory.setCustomEntryIdleTimeout`, `setCustomeEntryTimeToLive`

Example:

```
<region-attributes>
  <expiration-attributes
    timeout="60"
    action="local-destroy"/>
  <custom-expiry>
```

```
    <class-name>
      com.megaconglomerate.mypackage.MyClass
    </class-name>
  </custom-expiry>
</region-attributes>
```

Specify the Java class and its initialization parameters with the `<class-name>` and `<parameter>` sub-elements.

<region-idle-time>

Expiration setting that specifies how long the region can remain in the cache without anyone accessing it.

Note: To ensure reliable read behavior across the partitioned region, use `region-time-to-live` for region expiration instead of this setting.

Default: not set - no expiration of this type

API: `org.apache.geode.cache.RegionFactory.setRegionIdleTimeout`

Example:

```
<region-attributes statistics-enabled="true">
  <region-idle-time>
    <expiration-attributes
      timeout="3600"
      action="local-destroy"/>
  </region-idle-time>
</region-attributes>
```

<expiration-attributes>

Within the `entry-time-to-live` or `entry-idle-time` element, this element specifies the expiration rules for removing old region entries that you are not using. You can destroy or invalidate entries, either locally or across the cluster. Within the `region-time-to-live` or `region-idle-time` element, this element specifies the expiration rules for the entire region.

API: See APIs for `entry-time-to-live`, `entry-idle-time`, `region-time-to-live`, `region-idle-time`

<expiration-attributes> Attributes

Attribute	Description	Default
timeout	Number of seconds before a region or an entry expires. If timeout is not specified, it defaults to zero (which means no expiration).	0

Attribute	Description	Default
action	Action that should take place when a region or an entry expires. Select one of the following expiration actions:	invalidate

local-destroy	Removes the region or entry from the local cache, but does not distribute the removal operation to remote members. You cannot use this action on partitioned region entries.
destroy	Removes the region or entry completely from the cache. Destroy actions are distributed according to the region's distribution settings. Use this option when the region or entry is no longer needed for any application in the cluster.
invalidate	Default expiration action. Marks an entry or all entries in the region as invalid. Distributes the invalidation according to the region's scope. This is the proper choice when the region or the entry is no longer valid for any application in the cluster.
local-invalidate	Marks an entry or all entries in the region as invalid but does not distribute the operation. You cannot use this action on partitioned region entries. Local region invalidation is only supported for regions that are not configured as replicated regions.

Example:

```
<region-attributes
  statistics-enabled="true">
    <region-idle-time>
      <expiration-attributes
        timeout="60"
        action="local-destroy"/>
   </region-idle-time>
</region-attributes>
```

<custom-expiry>

Specifies the custom class that implements `org.apache.geode.cache.CustomExpiry`. You define this class in order to override the region-wide settings for specific entries. See Configure Data Expiration for an example.

Specify the Java class and its initialization parameters with the `<class-name>` and `<parameter>` sub-elements.

API: `org.apache.geode.cache.RegionFactory.setCustomEntryIdleTimeout`, `setCustomeEntryTimeToLive`

Example:

```
<region-attributes>
  <expiration-attributes
    timeout="60"
    action="local-destroy"/>
  <custom-expiry>
```

```
    <class-name>
      com.megaconglomerate.mypackage.MyClass
    </class-name>
  </custom-expiry>
</region-attributes>
```

Specify the Java class and its initialization parameters with the `<class-name>` and `<parameter>` sub-elements.

<entry-time-to-live>

Expiration setting that specifies how long the region's entries can remain in the cache without anyone accessing or updating them. See <expiration-attributes> for details.

Default: not set - no expiration of this type.

API: `org.apache.geode.cache.RegionFactory.setEntryTimeToLive`

Example:

```
<region-attributes
  statistics-enabled="true">
  <entry-time-to-live>
    <expiration-attributes
     timeout="60"
     action="local-destroy"/>
  </entry-time-to-live>
</region-attributes>
```

<expiration-attributes>

Within the `entry-time-to-live` or `entry-idle-time` element, this element specifies the expiration rules for removing old region entries that you are not using. You can destroy or invalidate entries, either locally or across the cluster. Within the `region-time-to-live` or `region-idle-time` element, this element specifies the expiration rules for the entire region.

API: See APIs for `entry-time-to-live`, `entry-idle-time`, `region-time-to-live`, `region-idle-time`

<expiration-attributes> Attributes

Attribute	Description	Default
timeout	Number of seconds before a region or an entry expires. If timeout is not specified, it defaults to zero (which means no expiration).	0

Attribute	Description		Default
action	Action that should take place when a region or an entry expires. Select one of the following expiration actions:		invalidate
	local-destroy	Removes the region or entry from the local cache, but does not distribute the removal operation to remote members. You cannot use this action on partitioned region entries.	
	destroy	Removes the region or entry completely from the cache. Destroy actions are distributed according to the region's distribution settings. Use this option when the region or entry is no longer needed for any application in the cluster.	
	invalidate	Default expiration action. Marks an entry or all entries in the region as invalid. Distributes the invalidation according to the region's scope. This is the proper choice when the region or the entry is no longer valid for any application in the cluster.	
	local-invalidate	Marks an entry or all entries in the region as invalid but does not distribute the operation. You cannot use this action on partitioned region entries. Local region invalidation is only supported for regions that are not configured as replicated regions.	

Example:

```
<region-attributes
  statistics-enabled="true">
   <entry-time-to-live>
      <expiration-attributes
        timeout="60"
        action="local-destroy"/>
   </entry-time-to-live>
</region-attributes>
```

<custom-expiry>

Specifies the custom class that implements `org.apache.geode.cache.CustomExpiry`. You define this class in order to override the region-wide settings for specific entries. See Configure Data Expiration for an example.

Specify the Java class and its initialization parameters with the `<class-name>` and `<parameter>` sub-elements.

API: `org.apache.geode.cache.RegionFactory.setCustomEntryIdleTimeout`, `setCustomeEntryTimeToLive`

Example:

```
<region-attributes>
    <expiration-attributes
       timeout="60"
       action="local-destroy">
      <custom-expiry>
```

```
        <class-name>
          com.megaconglomerate.mypackage.MyClass
        </class-name>
      </custom-expiry>
</region-attributes>
```

Specify the Java class and its initialization parameters with the `<class-name>` and `<parameter>` sub-elements.

<entry-idle-time>

Expiration setting that specifies how long the region's entries can remain in the cache without anyone accessing them. See <expiration-attributes> for details.

Note: To ensure reliable read behavior across the partitioned region, use `entry-time-to-live` for entry expiration instead of this setting.

API: `org.apache.geode.cache.RegionFactory.setEntryIdleTimeout`

Example:

```
<region-attributes
  statistics-enabled="true">
  <entry-idle-time>
    <expiration-attributes
      timeout="60"
      action="local-invalidate"/>
    </expiration-attributes>
  </entry-idle-time>
</region-attributes>
```

<expiration-attributes>

Within the `entry-time-to-live` or `entry-idle-time` element, this element specifies the expiration rules for removing old region entries that you are not using. You can destroy or invalidate entries, either locally or across the cluster. Within the `region-time-to-live` or `region-idle-time` element, this element specifies the expiration rules for the entire region.

API: See APIs for `entry-time-to-live`, `entry-idle-time`, `region-time-to-live`, `region-idle-time`

<expiration-attributes> Attributes

Attribute	Description	Default
timeout	Number of seconds before a region or an entry expires. If timeout is not specified, it defaults to zero (which means no expiration).	0

Attribute	Description		Default
action	Action that should take place when a region or an entry expires. Select one of the following expiration actions:		invalidate
	local-destroy	Removes the region or entry from the local cache, but does not distribute the removal operation to remote members. You cannot use this action on partitioned region entries.	
	destroy	Removes the region or entry completely from the cache. Destroy actions are distributed according to the region's distribution settings. Use this option when the region or entry is no longer needed for any application in the cluster.	
	invalidate	Default expiration action. Marks an entry or all entries in the region as invalid. Distributes the invalidation according to the region's scope. This is the proper choice when the region or the entry is no longer valid for any application in the cluster.	
	local-invalidate	Marks an entry or all entries in the region as invalid but does not distribute the operation. You cannot use this action on partitioned region entries. Local region invalidation is only supported for regions that are not configured as replicated regions.	

Example:

```
<region-attributes
  statistics-enabled="true">
   <entry-idle-time>
      <expiration-attributes
        timeout="60"
        action="local-destroy"/>
   </entry-idle-time>
</region-attributes>
```

<custom-expiry>

Specifies the custom class that implements `org.apache.geode.cache.CustomExpiry`. You define this class in order to override the region-wide settings for specific entries. See Configure Data Expiration for an example.

Specify the Java class and its initialization parameters with the `<class-name>` and `<parameter>` sub-elements.

API: `org.apache.geode.cache.RegionFactory.setCustomEntryIdleTimeout`, `setCustomeEntryTimeToLive`

Example:

```
<region-attributes>
    <expiration-attributes
       timeout="60"
       action="local-destroy">
      <custom-expiry>
```

```
        <class-name>
          com.megaconglomerate.mypackage.MyClass
        </class-name>
      </custom-expiry>
</region-attributes>
```

Specify the Java class and its initialization parameters with the `<class-name>` and `<parameter>` sub-elements.

<partition-attributes>

Defines the region as partitioned and controls partitioning behavior. This is set during the region creation in the first data store for the partitioned region.

Note: With the exception of `local-max-memory`, all members defining a partitioned region must use the same partition attribute settings.

API: `org.apache.geode.cache.RegionFactory.setPartitionAttributes`

<partition-attributes> Attributes

Attribute	Description	Default
colocated-with	The full name of a region to colocate with this region. The named region must exist before this region is created.	null
local-max-memory	Maximum megabytes of memory set aside for this region in the local member. This is all memory used for this partitioned region - for primary buckets and any redundant copies. This value must be smaller than the Java settings for the initial or maximum JVM heap. When the memory use goes above this value, VMware GemFire issues a warning, but operation continues. Besides setting the maximum memory to use for the member, this setting also tells VMware GemFire how to balance the load between members where the region is defined. For example, if one member sets this value to twice the value of another member's setting, VMware GemFire works to keep the ratio between the first and the second at two-to-one, regardless of how little memory the region consumes. This is a local parameter that applies only to the local member. A value of 0 disables local data caching.	90% (of local heap)
recovery-delay	Applies when `redundant-copies` is greater than zero. The number of milliseconds to wait after a member crashes before reestablishing redundancy for the region. A setting of -1 disables automatic recovery of redundancy after member failure.	-1
redundant-copies	Number of extra copies that the partitioned region must maintain for each entry. Range: 0-3. If you specify 1, this partitioned region maintains the original and one backup, for a total of two copies. A value of 0 disables redundancy.	0
startup-recovery-delay	Applies when `redundant-copies` is greater than zero. The number of milliseconds a newly started member should wait before trying to satisfy redundancy of region data stored on other members. A setting of -1 disables automatic recovery of redundancy after new members join.	0
total-max-memory	Maximum combined megabytes of memory to be used by all processes hosting this region for all copies, primary and redundant.	`Integer.MAX_VALUE`
total-num-buckets	Total number of buckets or data storage areas allotted for the entire partitioned region in the distributed cache. As data moves from one member to another, the entries in a bucket move as one unit. This value should be a prime number at least four times the number of data stores. More buckets increases overhead, however, especially when redundant-copies = 2 or 3 .	113

Example:

```
<region-attributes>
  <partition-attributes
   redundant-copies="1"
   total-num-buckets= "613"/>
</region-attributes>
```

<partition-resolver>

Describes a custom PartitionResolver for a region.

API: `org.apache.geode.cache.PartitionAttributesFactory.setPartitionResolver`

Specify the Java class and its initialization parameters with the `<class-name>` and `<parameter>` sub-elements. See <class-name> and <parameter>.

<partition-resolver> Attributes

Attribute	Description	Default
name	The name of this custom PartitionResolver.	

Example:

```
<region name="trades">
  <region-attributes>
      <partition-attributes>
          <partition-resolver name="TradesPartitionResolver">
              <class-name>myPackage.TradesPartitionResolver
              </class-name>
          </partition-resolver>
      </partition-attributes>
  </region-attributes>
</region>
```

<partition-listener>

Defines a custom PartitionListener for a partitioned region.

Specify the Java class and its initialization parameters with the `<class-name>` and `<parameter>` sub-elements. See <class-name> and <parameter>.

API: `org.apache.geode.cache.PartitionAttributesFactory.PartitionListener`

Example:

```
<partition-attributes redundant-copies="1">
     <partition-listener>
         <class-name>com.myCompany.ColocatingPartitionListener</class-name.
          <parameter name="viewRegions">
              <string>/customer/ViewA,/customer/ViewB</string>
          </parameter>
     </partition-listener>
 </partition-attributes>
```

<fixed-partition-attributes>

Describes a partition in a Fixed Partitioned Region.

API: `org.apache.geode.cache.PartitionAttributesFactory.addFixedPartitionAttributes`

<fixed-partition-attributes> Attributes

Attribute	Description	Default
partition-name	The name of this fixed partition.	
is-primary	Set to true if this partition is the primary partition.	false
num-buckets	The number of buckets assigned to this partition.	

Example:

```
<cache>
   <region name="Trades">
      <region-attributes>
         <partition-attributes
          redundant-copies="1">
          <partition-resolver name="QuarterFixedPartitionResolver">
             <fixed-partition-attributes
               partition-name="Q1"
               is-primary="true"/>
             <fixed-partition-attributes
               partition-name="Q3"
               is-primary="false"
               num-buckets="6"/>
         </partition-attributes>
      </region-attributes>
   </region>
</cache>
```

<membership-attributes>

Establishes reliability requirements and behavior for a region. Use this to configure the region to require one or more membership roles to be running in the system for reliable access to the region. You can set up your own roles, such as producer or backup, specifying each role as a string. Membership attributes have no effect unless one or more required roles are specified.

API: `org.apache.geode.cache.RegionFactory.setMembershipAttributes`

<membership-attributes> Attributes

Attribute	Description	Default
loss-action	Set one of the following values to specify how access to the region is affected when one or more required roles are lost. full-access: Access to the region is unaffected when required roles are missing. limited-access: Only local access to the region is allowed when required roles are missing. no-access: The region is unavailable when required roles are missing. reconnect: Loss of required roles causes the entire cache to be closed.	no_access
resumption-action	Specifies how the region is affected by resumption of reliability when one or more missing required roles return to the distributed membership. Set one of the following values: none: No special action takes place when reliability resumes. reinitialize: Resumption of reliability causes the region to be cleared of all data and replicated regions will do a new getInitialImage operation to repopulate the region.	reinitialize

Example:

```
<!-- If there is no "producer" member
  running, do not allow access to the region -->
<region-attributes>
  <membership-attributes
    loss-action="no-access"
    resumption-action="none">
    <required-role
      name="producer">
    </required-role>
  </membership-attributes>
</region-attributes>
```

<required-role>

Specifies a role that is required for reliable access to the region.

API: `org.apache.geode.cache.MembershipAttributes`

<required-role> Attributes

Attribute	Description	Default
name	The name of the required role.	

Example:

```
<membership-attributes
  loss-action="no-access"
  resumption-action="none">
    <required-role name="producer"/>
</membership-attributes>
```

<subscription-attributes>

Specifies subscriber requirements and behavior for the region. There is one subscription attribute, `interest-policy`, that defines which distributed entry events are delivered to the local region.

Note: The interest policy determines which events are delivered, but the `data-policy` determines how the events are applied to the cache.

API: `org.apache.geode.cache.RegionFactory.setSubscriptionAttributes`

<subscription-attributes> Attributes

Attribute	Description	Default
interest-policy	The two `interest-policy` options are:	cache-content
	cache-content: The default, registers interest in events only for entries that are already in the local region. For partitioned regions, the local member must hold the primary copy of the entry's data.	
	all: Registers interest in events for all entries that are anywhere in the distributed or partitioned region, regardless of whether they are already present in the local cache.	

Example:

```
<region-attributes>
  <subscription-attributes
   interest-policy="all"/>
</region-attributes>
```

<cache-loader>

An event-handler plug-in that allows you to program for cache misses. At most, one cache loader can be defined in each member for the region. For distributed regions, a cache loader may be invoked remotely from other members that have the region defined. When an entry get results in a cache miss in a region with a cache loader defined, the loader's load method is called. This method is usually programmed to retrieve data from an outside data source, but it can do anything required by your application.

For partitioned regions, if you want to have a cache loader, install an instance of the cache loader in every data store. Partitioned regions support partitioned loading, where each cache loader loads only the data entries in the local member. If data redundancy is configured, data is loaded only if the local member holds the primary copy.

API: `org.apache.geode.cache.RegionFactory.setCacheLoader`

Example:

```
<region-attributes>
  <cache-loader>
    <class-name>quickstart.SimpleCacheLoader</class-name>
  </cache-loader>
</region-attributes>
```

<cache-writer>

An event-handler plug-in that allows you to receive before-event notification for changes to the region and its entries. It also has the ability to cancel events. At most, one cache writer can be defined in each member for the region. A cache writer may be invoked remotely from other members that have the region defined.

API: `org.apache.geode.cache.RegionFactory.setCacheWriter`

Example:

```
<region-attributes>
  <cache-writer>
    <class-name>quickstart.SimpleCacheWriter</class-name>
  </cache-writer>
</region-attributes>
```

<cache-listener>

An event-handler plug-in that receives after-event notification of changes to the region and its entries. Any number of cache listeners can be defined for a region in any member. VMware GemFire offers several listener types with callbacks to handle data and process events. Depending on the `data-policy` and the `interest-policy` subscription attributes, a cache listener may receive only events that originate in the local cache, or it may receive those events along with events that originate remotely.

Specify the Java class for the cache listener and its initialization parameters with the `<class-name>` and `<parameter>` sub-elements. See <class-name> and <parameter>.

API: `org.apache.geode.cache.RegionFactory.addCacheListener`

Example:

```
<region-attributes>
  <cache-listener>
    <class-name>
      quickstart.SimpleCacheListener
    </class-name>
  </cache-listener>
</region-attributes>
```

<compressor>

A compressor registers a custom class that extends `Compressor` to support compression on a region.

Example:

```
 ...
<compressor>
<class-name>
            <parameter>
               <string>
               <declarable>
</compressor>
...
```

<eviction-attributes>

Specifies whether and how to control a region's size. Size is controlled by removing least recently used (LRU) entries to make space for new ones. This may be done through destroy or overflow actions. You can configure your region for lru-heap-percentage with an eviction action of local-destroy using stored region attributes.

Default: Uses the lru-entry-count algorithm.

API: `org.apache.geode.cache.RegionFactory.setEvictionAttributes`

Example:

```
<region-attributes>
  <eviction-attributes>
    <lru-entry-count
      maximum="1000"
      action="overflow-to-disk"/>
  </eviction-attributes>
</region-attributes
```

<lru-entry-count>

Using the maximum attribute, specifies maximum region capacity based on entry count.

<lru-entry-count> Attributes

Attribute	Description		Default
action	Set one of the following eviction actions:		local-destroy
	local-destroy	Removes the entry from the local cache, but does not distribute the removal operation to remote members. This action can be applied to an entry in a partitioned region, but is not recommended if redundancy is enabled (redundant-copies > 0), as it introduces inconsistencies between the redundant buckets. When applied to an entry in a replicated region, VMware GemFire silently changes the region type to "preloaded" to accommodate the local modification.	
	overflow-to-disk	The entry's value is overflowed to disk and set to null in memory. The entry's key is retained in the cache.	

Attribute	Description	Default
maximum	The maximum number of entries allowed in a region.	

<lru-heap-percentage>

Runs evictions when the VMware GemFire resource manager says to. The manager orders evictions when the total cache size is over the heap or off-heap percentage limit specified in the resource manager configuration. You can declare a Java class that implements the ObjectSizer interface to measure the size of objects in the Region.

Specify the Java class and its initialization parameters with the `<class-name>` and `<parameter>` sub-elements. See <class-name> and <parameter>.

<lru-heap-percentage> Attributes

Attribute	Description	Default
action	Set one of the following eviction actions:	local-destroy
	local-destroy: Removes the entry from the local cache, but does not distribute the removal operation to remote members. This action can be applied to an entry in a partitioned region, but is not recommended if redundancy is enabled (redundant-copies > 0), as it introduces inconsistencies between the redundant buckets. When applied to an entry in a replicated region, VMware GemFire silently changes the region type to "preloaded" to accommodate the local modification.	
	overflow-to-disk: The entry's value is overflowed to disk and set to null in memory. The entry's key is retained in the cache.	

<lru-memory-size>

Using the maximum attribute, specifies maximum region capacity based on the amount of memory used, in megabytes. You can declare a Java class that implements the ObjectSizer interface to measure the size of objects in the Region.

Specify the Java class and its initialization parameters with the `<class-name>` and `<parameter>` sub-elements. See <class-name> and <parameter>.

<lru-memory-size> Attributes

Attribute	Description	Default
action	Set one of the following eviction actions: local-destroy: Removes the entry from the local cache, but does not distribute the removal operation to remote members. This action can be applied to an entry in a partitioned region, but is not recommended if redundancy is enabled (redundant-copies > 0), as it introduces inconsistencies between the redundant buckets. When applied to an entry in a replicated region, VMware GemFire silently changes the region type to "preloaded" to accommodate the local modification. overflow-to-disk: The entry's value is overflowed to disk and set to null in memory. The entry's key is retained in the cache.	local-destroy
maximum	The maximum amount of memory used in the region, in megabytes.	

<jndi-bindings>

Specifies the binding for a data-source used in transaction management. See Configuring Database Connections Using JNDI.

Example:

```
<jndi-bindings>
      <jndi-binding type="XAPooledDataSource"
    jndi-name="newDB2trans"
    init-pool-size="20"
    max-pool-size="100"
    idle-timeout-seconds="20"
    blocking-timeout-seconds="5"
    login-timeout-seconds="10"
    xa-datasource-class="org.apache.derby.jdbc.EmbeddedXADataSource"
    user-name="mitul"
    password="encrypted(83f0069202c571faf1ae6c42b4ad46030e4e31c17409e19a)">
         <config-property>
          <config-property-name>Description</config-property-name>
          <config-property-type>java.lang.String</config-property-type>
          <config-property-value>pooled_transact</config-property-value>
       </config-property>
          <config-property>
             <config-property-name>DatabaseName</config-property-name>
             <config-property-type>java.lang.String</config-property-type>
             <config-property-value>newDB</config-property-value>
          </config-property>
          <config-property>
             <config-property-name>CreateDatabase</config-property-name>
             <config-property-type>java.lang.String</config-property-type>
             <config-property-value>create</config-property-value>
          </config-property>
       . . .
      </jndi-binding>
   </jndi-bindings>
```

<jndi-binding>

For every datasource that is bound to the JNDI tree, there should be one `<jndi-binding>` element. This element describes the property and the configuration of the datasource. VMware GemFire uses the attributes of the `<jndi-binding>` element for configuration. Use the `<config-property>` element to configure properties for the datasource.

We recommend that you set the username and password with the `user-name` and `password` jndi-binding attributes rather than using the `<config-property>` element.

<jndi-binding> Attributes

Attribute	Description	Default
blocking-timeout-seconds	The number of seconds that a connection remains associated with a transaction. If this value is exceeded, the connection is disassociated from the transaction.	120
conn-pooled-datasource-class	Java class used for the `PooledDataSource` type.	
connection-url	URL for connecting to the datasource. **Note:** If you are connecting to a JCA data source driver that implements XA transactions (where the jndi-binding type is **XAPooledDataSource**), do not use this attribute. Instead, define configuration properties for your database. See <config-property> for an example.	
idle-timeout-seconds	The maximum number of seconds that a connection can remain idle in a pool. When this threshold is reached, the connection is removed.	600
init-pool-size	The initial pool size of a PooledConnection (an XAConnection or a non-XAConnection).	10
jdbc-driver-class	Java class used for the `SimpleDataSource` type.	
jndi-name	The `jndi-name` attribute is the key binding parameter. If the value of jndi-name is a DataSource, it is bound as java:/myDatabase, where myDatabase is the name you assign to your data source. If the data source cannot be bound to JNDI at runtime, VMware GemFire logs a warning.	

Attribute	Description	Default
login-timeout-seconds	The maximum number of seconds for which a thread seeking a connection from a connection pool may be blocked. If the thread is unable to obtain connection in the stipulated time, a PoolException is thrown If a connection is available the thread returns immediately. If an existing connection is not available and the maximum number of connections in the pool has not been reached, a new connection is created and the thread returns immediately with the connection. If a connection is not available, the thread blocks for the specified time while waiting for an available connection.	30
managed-conn-factory-class	If the Resource Adapter is of type ManagedDataSource, this class becomes the source of the PooledConnection. (This class interface complies with the J2CA Java 2 Connector Architecture.)	
max-pool-size	The maximum size of the PooledConnection.	30
password	Password to access the datasource.	

Attribute	Description	Default
transaction-type	When the `type` attribute is set to `ManagedDataSource`, specifies the type of transaction. Set one of the following `transaction-type`s: XA Transaction — Select this option when you want to use aManagedConnection interface with a Java Transaction Manager to define transaction boundaries. This option allows a ManagedDataSource to participate in a transaction with a VMware GemFire cache. No Transaction — No transactional behavior is used. Local Transaction — Select this option when using a ManagedDataSource that is not managed by the Java Transaction manager.	none

Attribute	Description	Default
type	Set one of the following types: XAPooledDataSource — Pooled SQL connections. For this type, you must also set the `xa-datasource-class` attribute. ManagedDataSource — JNDI binding type for the J2EE Connector Architecture (JCA). ManagedConnectionFactory. For information about the ManagedConnection interface, see Interface ManagedConnection in the Oracle documentation. PooledDataSource — Pooled SQL connections. For this type, you must also set the `conn-pooled-datasource-class` attribute. SimpleDataSource — Single SQL connection. No pooling of SQL connections is done. Connections are generated on the fly and cannot be reused. For this type, you must also set the `jdbc-driver-class` attribute.	none
user-name	User name to access to the datasource.	
xa-datasource-class	Java class used for the `XAPooledDataSource` type.	

<config-property>

A configuration property of the datasource. Use the sub-elements to identify the name, datatype, and value of the property.

Default:

Example:

```
<config-property>
     <config-property-name>DatabaseName</config-property-name>
     <config-property-type>java.lang.String</config-property-type>
     <config-property-value>newDB</config-property-value>
</config-property>
```

Configuration properties vary depending on the database vendor. See Configuring Database Connections Using JNDI for examples of different configuration property configurations.

<config-property-name>

The name of this datasource property.

<config-property-type>

The data type of this datasource property.

<config-property-value>

The value of this datasource property.

<region>

Defines a region in the cache. See <region-attributes> for more details on configuring regions. You can specify zero or more subregions within a region. See Create and Access Data Subregions for restrictions on creating subregions. For example, you cannot create a partitioned subregion.

Default:

API: `org.apache.geode.cache.RegionFactory` or `org.apache.geode.cache.ClientRegionFactory`

<region> Attributes

Attribute	Description	Default
name	Specify the name for the region. See Region Management for details.	
refid	Used to apply predefined attributes to the region being defined. If the nested “region-attributes” element has its own “refid”, then it will cause the “refid” on the region to be ignored. The “refid” region attribute can be set to the name of a RegionShortcut or a ClientRegionShortcut. For more information, see Region Shortcuts and Custom Named Region Attributes and Storing and Retrieving Region Shortcuts and Custom Named Region Attributes.	

Example:

```
<!--Using region shortcut-->
<region
  name="PartitionedRegion"
  refid="PARTITION_REDUNDANT">
...
</region>

<!-- Retrieving and storing attributes -->
<region-attributes
  id="myPartition"
  refid="PARTITION_REDUNDANT">
    <partition-attributes
   local-max-memory="512"/>
</region-attributes>
```

```
<!-- Attributes are retrieved and applied in the first region -->
<region name="PartitionedRegion1" refid="myPartition"/>
```

See <region-attributes> for a complete listing of region attributes.

<index>

Describes an index to be created on a region. The index node, if any, should all come immediately after the “region-attributes” node. The “name” attribute is a required field which identifies the name of the index. See Working with Indexes for more information on indexes.

Default:

API: `org.apache.geode.cache.query.QueryService.createIndex, createKeyIndex, createHashIndex`

<index> Attributes

Attribute	Description	Default
name	Required. Name of the index.	
from-clause	Specifies the collection(s) of objects that the index ranges over. The from-clause must only contain one and only one region path.	
expression	Specifies the lookup value of the index.	
imports	String containing the imports used to create the index. String should be specified in the query language syntax with each import statement separated by a semicolon. The imports statement provides packages and classes used in variable typing in the indexed and FROM expressions.	
key-index	True or false. Whether the index should be a key index. If true, the region key specified in the indexed expression is used to evaluate queries	
type	Possible values are “hash” or “range”.	range

Example:

```
<region name=exampleRegion>
 <region-attributes . . . >
 </region-attributes>
 <index
  name="myIndex"
  from-clause="/exampleRegion"
  expression="status"/>
 <index
  name="myKeyIndex"
  from-clause="/exampleRegion"
  expression="id" key-index="true"/>
 <index
  name="myHashIndex"
  from-clause="/exampleRegion p"
  expression="p.mktValue" type="hash"/>
 ...
</region>
```

<lucene:index>

Describes a Lucene index to be created on a region. The `lucene` namespace and the scoping operator (`:`) must be specified, as the VMware GemFire `cache` namespace also defines an `index` element (for OQL indexes).

API: `org.apache.geode.cache.lucene` package

<lucene:index> Attributes | Attribute | Description | Default | |————-|
———
——|
———| | name | Required. Name of the Lucene index. | |

Example:

```
<cache
    xmlns="http://geode.apache.org/schema/cache"
    xmlns:lucene="http://geode.apache.org/schema/lucene"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://geode.apache.org/schema/cache
        http://geode.apache.org/schema/cache/cache-1.0.xsd
        http://geode.apache.org/schema/lucene
        http://geode.apache.org/schema/lucene/lucene-1.0.xsd"
    version="1.0">

    <region name="regionA" refid="PARTITION">
        <lucene:index name="myIndex">
            <lucene:field name="x" />
            <lucene:field name="y" />
        </lucene:index>
    </region>
</cache>
```

<lucene:field>

Describes a field to be included in a Lucene index. Including the `lucene` namespace and the scoping operator (`:`) clarifies, but is not required.

API: `org.apache.geode.cache.lucene` package

<lucene:field&gt Attributes | Attribute | Description | Default | |————-|————-|———| | name | Required. A string that defines the name of the field. If a single field is defined by the value `"__REGION_VALUE_FIELD"`, then the entire value is used as a single field. | | | analyzer | A string that provides the path to the analyzer to use for this field. A value of `"null"` uses the default analyzer. | `"null"` |

Example:

```
<region name="dataregion" refid="PARTITION_REDUNDANT">
   <lucene:index name="full_value_index">
     <lucene:field name="__REGION_VALUE_FIELD"/>
   </lucene:index>
</region>
```

<entry>

An “entry” element describes an entry to be added to a region. Note that if an entry with the given key already exists in the region, it will be replaced.

Default:

API: `org.apache.geode.cache.Region.create`, `put`, `get`, `putAll`, `getAll`

Example:

```
<region ...>
 <region-attributes ...>
   ...
 </region-attributes>
 <entry>
   <key><string>MyKey</string></key>
   <value><string>MyValue</string></value>
 </entry>
</region>
```

<key>

Required. Describes the key in a region entry. A key can contain either a <string> or a <declarable> sub-element.

<string>

Specifies a String to be placed in a Region entry.

Example:

```
<region ...>
 <region-attributes ...>
   ...
 </region-attributes>
 <entry>
   <key><string>MyKey</string></key>
   <value><string>MyValue</string></value>
 </entry>
</region>
```

<declarable>

Specifies a Declarable object to be placed in a Region entry.

Specify the Java class and its initialization parameters with the `<class-name>` and `<parameter>` sub-elements.

API: `Declarable`

Example:

```
<parameter name="cacheserver">
  <declarable>
```

```
    <class-name>org.apache.geode.addon.cache.CacheServerInitializer</class-name>
    <parameter name="system.property.prefix">
    <string>cacheserver</string>
    </parameter>
  </declarable>
</parameter>
```

<value>

Required. Describes the value of a region entry. A `<value>` can contain either a `<string>` or a `<declarable>` sub-element.

<string>

Specifies a String to be placed in a Region entry.

Example:

```
<region ...>
 <region-attributes ...>
   ...
 </region-attributes>
 <entry>
   <key><string>MyKey</string></key>
   <value><string>MyValue</string></value>
 </entry>
</region>
```

<declarable>

Specifies a Declarable object to be placed in a Region entry.

Specify the Java class and its initialization parameters with the `<class-name>` and `<parameter>` sub-elements.

API: `Declarable`

Example:

```
<parameter name="cacheserver">
  <declarable>
    <class-name>org.apache.geode.addon.cache.CacheServerInitializer</class-name>
    <parameter name="system.property.prefix">
    <string>cacheserver</string>
    </parameter>
  </declarable>
</parameter>
```

<region>

When nested within a `<region>` element, defines a subregion. See Create and Access Data Subregions for restrictions on creating subregions. For example, you cannot create a partitioned subregion.

See <region>

<function-service>

Configures the behavior of the function execution service.

Example:

```
<cache>
    ...
    </region>
<function-service>
  <function>
    <class-name>com.myCompany.tradeService.cache.func.TradeCalc</class-name>
  </function>
</function-service>
```

<function>

Defines a function for registration in the function service

Specify the Java class for the function and its initialization parameters with the `<class-name>` and `<parameter>` sub-elements. See <class-name> and <parameter>.

Default:

API: `org.apache.geode.cache.execute.FunctionService`

Example:

```
<function>
  <class-name>
    com.myCompany.tradeService.cache.func.TradeCalc
  </class-name>
</function>
```

<resource-manager>

A memory monitor that tracks cache size as a percentage of total heap or off-heap memory and controls size by restricting access to the cache and prompting eviction of old entries from the cache. For tenured heap, used in conjunction with settings for JVM memory and Java garbage collection. For off-heap memory, used with the off-heap memory manager.

API: `org.apache.geode.cache.control.ResourceManager`

<resource-manager> Attributes

Attribute	Description	Default
critical-heap-percentage	Percentage of heap at or above which the cache is considered in danger of becoming inoperable due to garbage collection pauses or out of memory exceptions. Only one change to this attribute or critical heap percentage will be allowed at any given time and its effect will be fully realized before the next change is allowed. This feature requires additional VM flags to perform properly. See `setCriticalHeapPercentage()` for details.	0
eviction-heap-percentage	Set the percentage of heap at or above which the eviction should begin on regions configured for HeapLRU eviction. Changing this value may cause eviction to begin immediately. • The default is 0, if no region is configured with heap eviction • If `critical-heap-percentage` is set to a non-zero value, 5% less than that value. • 80%, if `critical-heap-percentage` is not configured.	0
critical-off-heap-percentage	Percentage of off-heap memory at or above which the cache is considered in danger of becoming inoperable due to garbage collection pauses or out of memory exceptions.	0
eviction-off-heap-percentage	Set the percentage of off-heap memory at or above which the eviction should begin on Regions configured for HeapLRU eviction. • The default is 0, if no region is configured with heap eviction • If `critical-off-heap-percentage` is set to a non-zero value, 5% less than that value. • 80%, if `critical-off-heap-percentage` is not configured.	0

Example:

```
<cache>
...
   <resource-manager
      critical-heap-percentage="99.9"
      eviction-heap-percentage="85"/>
...
</cache>
```

<serialization-registration>

Set of serializer or instantiator tags to register customer DataSerializer extensions or DataSerializable implementations respectively.

Example:

```
<serialization-registration>
    <instantiator id="30">
       <class-name>com.package.MyClass</class-name>
    </instantiator>
</serialization-registration>
```

<serializer>

Allows you to configure the DataSerializer for this VMware GemFire member. It registers a custom class which extends DataSerializer to support custom serialization of non-modifiable object types inside VMware GemFire.

Specify the Java class for the `DataSerializer` and its initialization parameters with the `<class-name>` sub-element.

API: You can also register a `DataSerializer` by using the `org.apache.geode.DataSerializer.register` API. Use the `org.apache.geode.Instantiator` API to register a `DataSerializable` implementation.

<instantiator>

An Instantiator registers a custom class which implements the `DataSerializable` interface to support custom object serialization inside VMware GemFire.

Specify the Java class and its initialization parameters with the `<class-name>` sub-element.

API: `DataSerializable`

You can also directly specify `<instantiator>` as a sub-element of `<cache>`. Use the `org.apache.geode.Instantiator` API to register a `DataSerializable` implementation as the serialization framework for the cache. The following table lists the attribute that can be specified for an `<instantiator>`.

<instantiator> Attributes

Attribute	Description	Default
id	Required. ID that the Instantiator should associate with the `DataSerializable` type.	

<backup>

Defines additional files or directories that should be backed up when the system wide backup command is invoked. Disk stores with persistent data are automatically backed up and do not need to be listed with this element.

Example:

```
<backup>./systemConfig/gf.jar</backup>
<backup>/users/jpearson/gfSystemInfo/myCustomerConfig.doc</backup>
```

<initializer>

Used to specify a callback class (and optionally its parameters) that will be run after the cache is initialized. This element can be specified for both server and client caches.

Specify the Java class and its initialization parameters with the `<class-name>` and `<parameter>` sub-elements. See <class-name> and <parameter>.

Default:

API: `Declarable`

Example:

```
<initializer>
   <class-name>MyInitializer</class-name>
      <parameter name="members">
         <string>2</string>
      </parameter>
</initializer>
```

<declarable>

Specifies a Declarable object to be placed in a Region entry.

Specify the Java class and its initialization parameters with the `<class-name>` and `<parameter>` sub-elements.

API: `Declarable`

Example:

```
<parameter name="cacheserver">
  <declarable>
    <class-name>org.apache.geode.addon.cache.CacheServerInitializer</class-name>
    <parameter name="system.property.prefix">
    <string>cacheserver</string>
    </parameter>
  </declarable>
</parameter>
```

<class-name> and <parameter>

Specify the name of a Java class with the `<class-name>` sub-element.

Specify initialization parameters for the class using the `<parameter>` sub-element. Use the `name` attribute to specify the name of the parameter and specify its value in the content of the <string> sub-element or by specifying a Java class with the <declarable> sub-element.

Example:

The following transaction writer configuration example specifies a Java class named `com.company.data.MyTransactionWritet`. The class is initialized with a parameter named `URL` whose value is `jdbc:cloudscape:rmi:MyData`.

```
<transaction-writer>
       <class-name>com.company.data.MyTransactionWriter</class-name>
       <parameter name="URL">
         <string>jdbc:cloudscape:rmi:MyData</string>
       </parameter>
</transaction-writer>
```

<declarable>

Specifies a Declarable object to be placed in a Region entry.

Specify the Java class and its initialization parameters with the `<class-name>` and `<parameter>` sub-elements.

API: `Declarable`

Example:

```
<parameter name="cacheserver">
  <declarable>
    <class-name>org.apache.geode.addon.cache.CacheServerInitializer</class-name>
    <parameter name="system.property.prefix">
    <string>cacheserver</string>
    </parameter>
  </declarable>
</parameter>
```

<string>

Specifies a String to be placed in a Region entry.

Example:

```
<region ...>
 <region-attributes ...>
   ...
 </region-attributes>
 <entry>
   <key><string>MyKey</string></key>
   <value><string>MyValue</string></value>
 </entry>
</region>
```

<client-cache> Element Hierarchy

This section shows the hierarchy of `<client-cache>` element sub-elements that you use to configure VMware GemFire caches and clients.

For details, see <client-cache> Element Reference.

```
<client-cache>
   <cache-transaction-manager>
      <transaction-listener>
      <transaction-writer>
   <pool>
      <locator>
      <server>
   <disk-store>
      <disk-dirs>
         <disk-dir>
   <pdx>
      <pdx-serializer>
         <class-name>
         <parameter>
            <string>
            <declarable>
   <region-attributes>
      <key-constraint>
      <value-constraint>
      <region-time-to-live>
         <expiration-attributes>
            <custom-expiry>
               <class-name>
               <parameter>
                  <string>
                  <declarable>
      <region-idle-time>
         <expiration-attributes>
            <custom-expiry>
               <class-name>
               <parameter>
                  <string>
                  <declarable>
      <entry-time-to-live>
         <expiration-attributes>
            <custom-expiry>
               <class-name>
               <parameter>
                  <string>
                  <declarable>
      <entry-idle-time>
         <expiration-attributes>
            <custom-expiry>
               <class-name>
               <parameter>
                  <string>
                  <declarable>
      <cache-loader>
         <class-name>
            <parameter>
               <string>
               <declarable>
      <cache-writer>
         <class-name>
            <parameter>
               <string>
               <declarable>
      <cache-listener>
         <class-name>
            <parameter>
               <string>
               <declarable>
      <eviction-attributes>
         <lru-entry-count>
         <lru-heap-percentage>
            <class-name>
               <parameter>
                  <string>
                  <declarable>
         <lru-memory-size>
            <class-name>
               <parameter>
                  <string>
                  <declarable>
```

```
    <jndi-bindings>
       <jndi-binding>
          <config-property>
             <config-property-name>
             <config-property-type>
             <config-property-value>
    <region>
       <region-attributes>
       <index>
          <entry>
          <key>
             <string>
             <declarable>
          <value>
             <string>
             <declarable>
       <region>
    <function-service>
       <function>
          <class-name>
          <parameter>
             <string>
             <declarable>
    <resource-manager>
    <serialization-registration>
       <serializer>
          <class-name>
       <instantiator>
          <class-name>
    <initializer>
       <class-name>
       <parameter>
          <string>
          <declarable>
</client-cache>
```

<client-cache> Element Reference

This section documents all `cache.xml` elements that you use to configure VMware GemFire clients. All elements are sub-elements of the `<client-cache>` element.

For VMware GemFire server configuration, see <cache> Element Reference.

API: `org.apache.geode.cache.client.ClientCacheFactory` and `PoolFactory` interfaces.

<client-cache> Attributes

Attribute	Definition	Default
copy-on-read	Boolean indicating whether entry value retrieval methods return direct references to the entry value objects in the cache (false) or copies of the objects (true).	False

Example:

```
<client-cache>
  <pool
    name="client"
    subscription-enabled="true">
    <locator host="localhost" port="41111"/>
  </pool>
  <region-attributes
    id="clientAttributes"
    pool-name="client"
    refid="CACHING_PROXY"/>
  <region name="root">
    <region-attributes scope="local"/>
    <region name="cs_region">
      <region-attributes refid="clientAttributes"/>
    </region>
  </region>
</client-cache>
```

<cache-transaction-manager>

Specifies a transaction listener.

API: `CacheTransactionManager`

Example:

```
<client-cache search-timeout="60">
   <cache-transaction-manager>
     <transaction-listener>
       <class-name>com.company.data.MyTransactionListener</class-name>
       <parameter name="URL">
         <string>jdbc:cloudscape:rmi:MyData</string>
       </parameter>
     </transaction-listener>
     <transaction-listener>... </transaction-listener>
     <transaction-writer>
       <class-name>com.company.data.MyTransactionWriter</class-name>
       <parameter name="URL">
         <string>jdbc:cloudscape:rmi:MyData</string>
       </parameter>
       <parameter>
     </transaction-writer>
   </cache-transaction-manager> .. .
</client-cache>
```

<transaction-listener>

When a transaction ends, its thread calls the TransactionListener to perform the appropriate follow-up for successful commits, failed commits, or voluntary rollbacks.

Specify the Java class and its initialization parameters with the `<class-name>` and `<parameter>` sub-elements. See <class-name> and <parameter>.

<transaction-writer>

When you commit a transaction, a TransactionWriter can perform additional tasks, including cancelling the transaction.

Specify the Java class and its initialization parameters with the `<class-name>` and `<parameter>` sub-elements. See <class-name> and <parameter>.

<pool>

Use for client caches. Defines a client's server pool used to communicate with servers running in a different cluster.

API: `org.apache.geode.cache.client.PoolFactory`

<pool> Attributes

Attribute	Description	Default
free-connection-timeout	Amount of time a thread will wait to get a pool connection before timing out with an exception. This timeout keeps threads from waiting indefinitely when the pool's `max-connections` has been reached and all connections in the pool are in use by other threads.	10000
idle-timeout	Maximum time, in milliseconds, a pool connection can stay open without being used when there are more than `min-connections` in the pool. Pings over the connection do not count as connection use. If set to -1, there is no idle timeout.	5000
load-conditioning-interval	Amount of time, in milliseconds, a pool connection can remain open before being eligible for silent replacement to a less-loaded server.	300000 (5 minutes)
max-connections	Maximum number of pool connections the pool can create. If the maximum connections are in use, an operation requiring a client-to-server connection blocks until a connection becomes available or the `free-connection-timeout` is reached. If set to -1, there is no maximum. The setting must indicate a cap greater than `min-connections`. **Note**: If you need to use this to cap your pool connections, you should disable the pool attribute `pr-single-hop-enabled`. Leaving single hop enabled can increase thrashing and lower performance.	-1
min-connections	Minimum number of pool connections to keep available at all times. Used to establish the initial connection pool. If set to 0 (zero), no connection is created until an operation requires it. This number is the starting point, with more connections added later as needed, up to the `max-connection` setting. The setting must be an integer greater than or equal to 0.	1

Attribute	Description	Default
multiuser-authentication	Used for installations with security where you want to accommodate multiple users within a single client. If set to true, the pool provides authorization for multiple user instances in the same client application, and each user accesses the cache through its own `RegionService` instance. If false, the client either uses no authorization or just provides credentials for the single client process.	false
name	Name of this pool. Used in the client region pool-name to assign this pool to a region in the client cache. **Note**: This is a required property with no default setting.	none
ping-interval	How often to communicate with the server to show the client is alive, set in milliseconds. Pings are only sent when the ping-interval elapses between normal client messages. **Note**: Set this lower than the server's `maximum-time-between-pings`.	10000
pr-single-hop-enabled	Setting used to improve access to partitioned region data in the servers. Indicates whether to use metadata about the partitioned region data storage locations to decide where to send some data requests. This allows a client to send a data operation directly to the server hosting the key. Without this, the client contacts any available server and that server contacts the data store. This is used only for operations that can be carried out on a server-by-server basis, like put, get, and destroy.	true
read-timeout	Maximum time, in milliseconds, for the client to wait for a response from a server.	10000
retry-attempts	Number of times to retry a client request before giving up. If one server fails, the pool moves to the next, and so on until it is successful or it hits this limit. If the available servers are fewer than this setting, the pool will retry servers that have already failed until it reaches the limit. If this is set to -1, the pool tries every available server once.	-1
server-group	Logical named server group to use from the pool. A null value uses the global server group to which all servers belong. **Note**: This is only used when the `locator` list is defined.	null
socket-buffer-size	Size for socket buffers from the client to the server. Default: 32768.	32768

Attribute	Description	Default
socket-connect-timeout	The number of milliseconds to wait before timing out on the client socket connect to a server or locator. A value of zero is interpreted as an infinite timeout. If zero, the connection will block until established or an error occurs.	59000
statistic-interval	Interval, in milliseconds, at which to send client statistics to the server. If set to -1, statistics are not sent.	-1
subscription-ack-interval	Time, in milliseconds, between messages to the primary server to acknowledge event receipt. **Note**: Used only when `subscription-redundancy` is not '0' (zero).	100
subscription-enabled	Boolean indicating whether the server should connect back to the client and automatically sends server-side cache update information. Any bind address information for the client is automatically passed to the server for use in the callbacks.	false
subscription-message- tracking-timeout	Time-to-live, in milliseconds, for entries in the client's message tracking list.	900000 (15 minutes)
subscription-redundancy	Number of servers to use as backup to the primary for highly available subscription queue management. If set to 0, none are used. If set to -1, all available servers are used.	0
subscription-timeout-multiplier	Number of missing server pings that trigger timeout of a subscription feed. A value of zero (the default) disables timeouts. A value of one or more times out the server connection after the specified number of ping intervals have elapsed. A value of one is not recommended.	0

Example:

```
<pool
   name="publisher"
   subscription-enabled="true">
     <locator
       host="myLocatorAddress1"
       port="12345"/>
     <locator
       host="myLocatorAddress2"
       port="45678"/>
</pool>
```

<locator>

Addresses and ports of the locators to connect to. You can define multiple locators for the pool.

Note: Provide a locator list or `server` list, but not both.

API: `org.apache.geode.distributed.LocatorLauncher`

<locator> Attributes

Attribute	Description	Default
host	Hostname of the locator	
port	Port number of the locator	

Example:

```
<pool ...>
<locator
       host="myLocatorHost"
       port="12345"/>
```

<server>

Addresses and ports of the servers to connect to.

Note: Provide a server list or `locator` list, but not both.

Default:

API: `org.apache.geode.distributed.ServerLauncher`

<server> Attributes

Attribute	Description	Default
host	Hostname of the server	
port	Port number of the server	

Example:

```
<pool ...>
   <server
       host="myServerHost"
       port="123456"/>
</pool>
```

<disk-store>

Defines a pool of one or more disk stores, which can be used by regions, and client subscription queues.

Default: The cache default disk store, named “DEFAULT”, is used when disk is used but no disk store is named.

API: `org.apache.geode.cache.DiskStore`

<disk-store> Attributes

Attribute	Description	Default
name	The name of the Disk Store.	

Attribute	Description	Default
auto-compact	Set to true to automatically compact the disk files.	
compaction-threshold	The threshold at which an oplog will become compactable. Until it reaches this threshold the oplog will not be compacted. The threshold is a percentage in the range 0 to 100.	
allow-force-compaction	Set to true to allow disk compaction to be forced on this disk store.	
max-oplog-size	The maximum size, in megabytes, of an oplog (operation log) file.	
time-interval	The number of milliseconds that can elapse before unwritten data is written to disk.	
write-buffer-size	The size of the write buffer that this disk store uses when writing data to disk. Larger values may increase performance but use more memory. The disk store allocates one direct memory buffer of this size.	
queue-size	Maximum number of operations that can be asynchronously queued to be written to disk.	
disk-usage-warning-percentage	Disk usage above this threshold generates a warning message. For example, if the threshold is set to 90%, then on a 1 TB drive falling under 100 GB of free disk space generates the warning. Set to "0" (zero) to disable.	90
disk-usage-critical-percentage	Disk usage above this threshold generates an error message and shuts down the member's cache. For example, if the threshold is set to 99%, then falling under 10 GB of free disk space on a 1 TB drive generates the error and shuts down the cache. Set to "0" (zero) to disable.	99

Example:

```
<disk-store
    name="DEFAULT"
    allow-force-compaction="true">
     <disk-dirs>
        <disk-dir>/export/thor/customerData</disk-dir>
        <disk-dir>/export/odin/customerData</disk-dir>
        <disk-dir>/export/embla/customerData</disk-dir>
     </disk-dirs>
</disk-store>
```

<disk-dirs>

An element of a disk store that defines a set of `<disk-dir>` elements.

<disk-dir>

Specifies a region or disk store's disk directory.

<disk-dir> Attributes

Attribute	Description	Default
dir-size	Maximum amount of space to use for the disk store, in megabytes.	214748364 (2 petabytes)

Example:

```
<disk-dir
    dir-size="20480">/host3/users/gf/memberA_DStore</disk-dir>
```

<pdx>

Specifies the configuration for the Portable Data eXchange (PDX) method of serialization.

API: `org.apache.geode.cache.CacheFactory.setPdxReadSerialized`, `setPdxDiskStore`, `setPdxPersistent`, `setPdxIgnoreUnreadFields` and `org.apache.geode.cache.ClientCacheFactory.setPdxReadSerialized`, `setPdxDiskStore`, `setPdxPersistent`, `setPdxIgnoreUnreadFields`

Attribute	Description	Default
read-serialized	Set it to true if you want PDX deserialization to produce a PdxInstance instead of an instance of the domain class.	
ignore-unread-fields	Set it to true if you do not want unread PDX fields to be preserved during deserialization. You can use this option to save memory. Set to true only in members that are only reading data from the cache.	
persistent	Set to true if you are using persistent regions. This causes the PDX type information to be written to disk.	
disk-store-name	If using persistence, this attribute allows you to configure the disk store that the PDX type data will be stored in. By default, the default disk store is used.	

Example:

```
<client-cache>
  <pdx persistent="true" disk-store-name="myDiskStore">
    <pdx-serializer>
      <class-name>
       org.apache.geode.pdx.ReflectionBasedAutoSerializer
      </class-name>
    <parameter name="classes">
      <string>com.company.domain.DomainObject</string>>
    </parameter>
  </pdx-serializer>
 </pdx>
  ...
</client-cache>
```

<pdx-serializer>

Allows you to configure the PdxSerializer for this VMware GemFire member.

Specify the Java class and its initialization parameters with the `<class-name>` and `<parameter>` sub-elements. See <class-name> and <parameter>.

Default:

API: `org.apache.geode.cache.CacheFactory.setPdxSerializer`

Example:

```
<client-cache>
  <pdx>
    <pdx-serializer>
     <class-name>com.company.ExamplePdxSerializer</class-name>
    </pdx-serializer>
  </pdx>
  ...
</client-cache>
```

<region-attributes>

Specifies a region attributes template that can be named (by `id`) and referenced (by `refid`) later in the `cache.xml` and through the API.

API: `org.apache.geode.cache.RegionFactory` or `org.apache.geode.cache.ClientRegionFactory`

<region-attributes> Attributes

Attribute	Description	Default

concurrency-level	Gives an estimate of the maximum number of application threads that will concurrently access a region entry at one time. This attribute does not apply to partitioned regions. This attribute helps VMware GemFire optimize the use of system resources and reduce thread contention. This sets an initial parameter on the underlying `java.util.ConcurrentHashMap` used for storing region entries.
	
	Note: Before you modify this, read the concurrency level description, then see the Java API documentation for `java.util.ConcurrentHashMap`.
	
	API: `setConcurrencyLevel`
	
	Example:
	
	`<region-attributes`
	`  concurrency-level="10">`
	`</region-attributes>`
		16 (threads)

Attribute	Description	Default
data-policy	Specifies how the local cache handles data for a region. This setting controls behavior such as local data storage and region initialization. **Note**: Configure the most common options using the region shortcuts, `RegionShortcut` and `ClientRegionShortcut`. The default `data-policy` of `normal` specifies local cache storage. The empty policy specifies no local storage. In the region shortcuts, empty corresponds to the settings with the string `PROXY`. You can use an empty region for event delivery to and from the local cache without the memory overhead of data storage. You can specify the following data policies: **empty**: No data storage in the local cache. The region always appears empty. Use this for event delivery to and from the local cache without the memory overhead of data storage - zero-footprint producers that only distribute data to others and zero-footprint consumers that only see events. To receive events with this, set the region's `subscription-attributes` `interest-policy` to all. **normal**: Data used locally (accessed with `get`s, stored with `put`s, etc.) is stored in the local cache. This policy allows the contents in the cache to differ from other caches. **partition**: Data is partitioned across local and remote caches using the automatic data distribution behavior of partitioned regions. Additional configuration is done in the `partition-attributes`. **replicate**: The region is initialized with the data from other caches. After initialization, all events for the distributed region are automatically copied into the local region, maintaining a replica of the entire distributed region in the local cache. Operations that would cause the contents to differ with other caches are not allowed. This is compatible with local `scope`, behaving the same as for normal. **persistent-partition**: Behaves the same as `partition` and also persists data to disk.	normal

Attribute	Description	Default

	<table><tr><td>persistent-replicate</td><td>Behaves the same as `replicate` and also persists data to disk.</td></tr><tr><td>preloaded</td><td>Initializes like a replicated region, then, once initialized, behaves like a normal region.</td></tr></table> **API:** `setDataPolicy` **Example**:
	```
	&lt;region-attributes
	  data-policy="replicate">
	</region-attributes&gt;
	```
	 This is similar to using a region shortcut with`refid`, however when you use the REPLICATE region shortcut, it automatically sets the region's scope to `distributed-ack`.
	```
	&lt;region-attributes
	  refid="REPLICATE">
	</region-attributes&gt;
	```
	 If you use `data-policy`, you must set the scope explicitly.	

Attribute	Description	Default

enable-async-conflation	For TCP/IP distributions between peers, specifies whether to allow aggregation of asynchronous messages sent by the producer member for the region. This is a special-purpose Boolean attribute that applies only when asynchronous queues are used for slow consumers. A false value disables conflation so that all asynchronous messages are sent individually. This special-purpose attribute gives you extra control over peer-to-peer communication between distributed regions using TCP/IP. This attribute does not apply to client/server communication or to communication using the UDP unicast or IP multicast protocols.
	
	Note: To use this attribute, the `multicast-enabled` region attribute `disable-tcp` in `gemfire.properties` must be false (the default for both). In addition, asynchronous queues must be enabled for slow consumers, specified with the `async`* gemfire properties.
	
	API: `setEnableAsyncConflation`
	
	Example:
	
	`<region-attributes`
	`  enable-async-conflation="false">`
	`</region-attributes>`
		true

enable-gateway	Determines whether the gateway is enabled for the region. When set to true, events in the region are sent to the defined gateway hubs. Used only with GemFire version 6.x gateway configurations. For GemFire 7.0 configuration, see the `gateway-sender-id` attribute of the `<region-attributes>` element.	false

enable-subscription-conflation	Boolean for server regions that specifies whether the server can conflate its messages to the client. A true value enables conflation.
	
	Note: The client can override this setting with the `conflate-events` property in its `gemfire.properties`.
	
	API: `setEnableSubscriptionConflation`
	
	Example:
	
	`<region-attributes`
	` enable-subscription-conflation="true">`
	` </region-attributes>`
		false

Attribute	Description	Default

gateway-sender-ids	Specifies one or more gateway sender IDs to use for distributing region events to remote VMware GemFire sites. Specify multiple IDs as a comma-separated list.
	
	API: `addGatewaySenderId`
	
	Example:
	
	`<region-attributes`
	`  gateway-sender-ids="nwsender,swsende`
	`r">`
	`</region-attributes>`
		not set

async-event-queue-ids	Specifies one or more asynchronous event queues to use for distributing region events an `AsyncEventListener` implementation (for example, for write-behind cache event handling). Specify multiple IDs as a comma-separated list.
	
	API: `addAsyncEventQueueId`
	
	Example:
	
	`<region-attributes`
	`  async-event-queue-ids="customerqueue,o`
	`rdersqueue">`
	`</region-attributes>`
		not set

hub-id	If the `enable-gateway` attribute is set to true, a comma-separated list of gateway hub IDs that receive events from the region. Used only with GemFire version 6.x gateway configurations. For GemFire 7.0 configuration, see the `gateway-sender-id` attribute of the `<region-attributes>` element.	null

id	Stores the region attribute settings in the cache with this identifier. Once stored, the attributes can be retrieved using the region attribute `refid`.
	
	API: `setId`
	
	Example:
	
	`<region-attributes`
	`  id="persistent-replicated">`
	`</region-attributes>`
		not set

Attribute	Description	Default

ignore-jta	Boolean that determines whether operations on this region participate in active JTA transactions or ignore them and operate outside of the transactions. This is primarily used in cache loaders, writers, and listeners that need to perform non-transactional operations on a region, such as caching a result set.
	
	API: `setIgnoreJTA`
	
	Example:
	
	`<region-attributes`
	`  ignore-jta="true">`
	`</region-attributes>`
		false

index-update-type	Specifies whether region indexes are maintained synchronously with region modifications, or asynchronously in a background thread. In the `cache.xml` file, this is set as a value, asynchronous or synchronous, assigned to the `index-update-type` region attribute. Set this through the API by passing a boolean to the `setIndexMaintenanceSynchronous` method.
	
	API: `setIndexMaintenanceSynchronous`
	
	Example:
	
	`<region-attributes`
	`  index-update-type="asynchronous">`
	`</region-attributes>`
		synchronous updates

initial-capacity	Together with the `load-factor` region attribute, sets the initial parameters on the underlying `java.util.ConcurrentHashMap` used for storing region entries.
	
	API: `setInitialCapacity`
	
	Example:
	
	`<region-attributes`
	`  initial-capacity="20">`
	`</region-attributes>`
		16

Attribute	Description	Default

is-lock-grantor	Determines whether this member defines itself as the lock grantor for the region at region creation time. This only specifies whether the member becomes lock grantor at creation and does not reflect the current state of the member's lock grantor status. The member's lock grantor status may change if another member subsequently defines the region with `is-lock-grantor` set to true. This attribute is only relevant for regions with `global scope`, as only they allow locking. It affects implicit and explicit locking.
	
	API: `setLockGrantor`
	
	Example:
	
	`<region-attributes`
	`  is-lock-grantor="true">`
	`</region-attributes>`
		false

load-factor	Together with the initial-capacity region attribute, sets the initial parameters on the underlying `java.util.ConcurrentHashMap` used for storing region entries. This must be a floating point number between 0 and 1, inclusive.
	
	Note: Before you set this attribute, read the discussion of initial capacity and load factor, then see the Java API documentation for `java.util.ConcurrentHashMap`.
	
	API: `setLoadFactor`
	
	Example:
	
	`<region-attributes`
	`  load-factor="0.85">`
	`</region-attributes>`
		.75

mirror-type	Deprecated	

multicast-enabled	Boolean that specifies whether distributed operations on a region should use multicasting. To enable this, multicast must be enabled for the cluster with the `mcast-port` `gemfire.properties` setting.
	
	API: `setMulticastEnabled`
	
	Example:
	
	`<region-attributes`
	`  multicast-enabled="true">`
	`</region-attributes>`
		false

Attribute	Description	Default

pool-name	Identifies the region as a client region and specifies the server pool the region is to use. The named pool must be defined in the client cache before the region is created. If this is not set, the region does not connect to the servers as a client region.
	
	API: `setPoolName`
	
	Examples:
	
	This declaration creates the region as a client region with a server pool named DatabasePool. This pool-name specification is required, as there are multiple pools in the client cache:
	
	`<client-cache>`
	`    <pool name="DatabasePool"`
	`   subscription-enabled="true">`
	`        ...`
	`    </pool>`
	`    <pool >`
	`   name="OtherPool"`
	`   subscription-enabled="true">`
	`        ...`
	`    </pool>`
	`    <region ...`
	`        <region-attributes`
	`          pool-name="DatabasePool">`
	`        </region-attributes>`
	`        ...`
	
	This declaration creates the region as a client region assigned the single pool that is defined for the client cache. Here the pool-name specification is implied to be the only pool that exists in the cache:
	
	`<client-cache>`
	`    <pool`
	`   name="publisher"`
	`   subscription-enabled="true">`
	`        ...`
	`    </pool>`
	`    <region`
	`   name="myRegion"`
	`   refid="CACHING_PROXY">`
	`    </region>`
	`</client-cache>`
		not set

Attribute	Description	Default

disk-store-name	Assigns the region to the disk store with this name from the disk stores defined for the cache. Persist region data to disk by defining the region as persistent using the Shortcut Attribute Options or data-policy settings. Overflow data to disk by implementing LRU eviction-attributes with an action of overflow to disk. Each disk store defines the file system directories to use, how data is written to disk, and other disk storage maintenance properties. In addition, the `disk-synchronous` region attribute specifies whether writes are done synchronously or asynchronously.
	
	API: `setDiskStoreName`
	
	Example:
	
	`<region-attributes`
	`  disk-store-name="myStoreA" >`
	`</region-attributes>`
		null

disk-synchronous	For regions that write to disk, boolean that specifies whether disk writes are done synchronously for the region.
	
	API: `setDiskSynchronous`
	
	Example:
	
	`<region-attributes`
	`  disk-store-name="myStoreA"`
	`  disk-synchronous="true">`
	`</region-attributes>`
		true

refid	Retrieves region shortcuts and user-defined named region attributes for attributes initialization
	
	API: `setRefId`
	
	Example:
	
	`<region-attributes`
	`  refid="persistent-replicated">`
	`  <!-- Override any stored`
	`    attribute settings that you`
	`    need to ... -->`
	`</region-attributes>`
		not set

Attribute	Description	Default
scope	Definition: Determines how updates to region entries are distributed to the other caches in the cluster where the region and entry are defined. Scope also determines whether to allow remote invocation of some of the region's event handlers, and whether to use region entry versions to provide consistent updates across replicated regions. **Note**: You can configure the most common of these options with VMware GemFire region shortcuts in `RegionShortcut` and `ClientRegionShortcut`. **Note**: Server regions that are not partitioned must be replicated with `distributed-ack` or `global` scope. The region shortcuts that specify `REPLICATE` have `distributed-ack` scope. Set one of the following scope values: local — No distribution. The region is visible only to threads running inside the member. distributed-no-ack — Events are distributed to remote caches with no acknowledgement required. distributed-ack — Events are distributed to remote caches with receipt acknowledgement required. Region entry versions are used to provide consistent updates across members of the cluster. global — Events are distributed to remote caches with global locking to ensure distributed cache consistency. **API:** `setScope` **Example:** `<region-attributes scope="distributed-ack"> </region-attributes>`	distributed-no-ack

```
<region-attributes
  scope="distributed-ack">
</region-attributes>
```

Attribute	Description	Default

statistics-enabled	Boolean specifying whether to gather statistics on the region. Must be true to use expiration on the region. VMware GemFire provides a standard set of statistics for cached regions and region entries, which give you information for fine-tuning your cluster. Unlike other VMware GemFire statistics, statistics for local and distributed regions are not archived and cannot be charted. They are kept in instances of `org.apache.geode.cache.CacheStatistics` and made available through the region and its entries through the `Region.getStatistics` and `Region.Entry.getStatistics` methods.
	
	API: `setStatisticsEnabled`
	
	Example:
	
	`<region-attributes`
	`  statistics-enabled="true">`
	`</region-attributes>`
		false

cloning-enabled	Determines how `fromDelta` applies deltas to the local cache for delta propagation. When true, the updates are applied to a clone of the value and then the clone is saved to the cache. When false, the value is modified in place in the cache.
	
	API: `setCloningEnabled`
	
	Example:
	
	`<region-attributes`
	`  cloning-enabled="true">`
	`</region-attributes>`
		false

Attribute	Description	Default

concurrency-checks-enabled	Determines whether members perform checks to provide consistent handling for concurrent or out-of-order updates to distributed regions. See Consistency for Region Updates.
	
	Note: Applications that use a `client-cache` may want to disable concurrency checking in order to see all events for a region. VMware GemFire server members can continue using concurrency checks for the region, but they will pass all events to the client cache. This configuration ensures that the client sees all events, but it does not prevent the client cache from becoming out-of-sync with the server cache.
	
	API: `setConcurrencyChecksEnabled`
	
	Example:
	
	`<region-attributes`
	` concurrency-checks-enabled="true">`
	`</region-attributes>`
		true

off-heap	Specifies that the region uses off-heap memory to store entry values, including values for region entries and queue entries. The region will still use heap memory for everything else, such as entry keys and the ConcurrentHashMap.
	
	API: `setOffHeap`
	
	Example:
	
	`<region-attributes`
	`  off-heap="true">`
	`</region-attributes>`
		false

<key-constraint>

Defines the type of object to be allowed for the region entry keys. This must be a fully-qualified class name. The attribute ensures that the keys for the region entries are all of the same class. If key-constraint is not used, the region's keys can be of any class. This attribute, along with value-constraint, is useful for querying and indexing because it provides object type information to the query engine.

Note: Set the constraint in every cache where you create or update the region entries. For client/server installations, match constraints between client and server and between clusters. The constraint is only checked in the cache that does the entry `put` or `create` operation. To avoid deserializing the object, the constraint is not checked when the entry is distributed to other caches.

Default: not set

API: `org.apache.geode.cache.RegionFactory.setKeyConstraint`

Example:

```
<region-attributes>
  <key-constraint>
   java.lang.String
  </key-constraint>
</region-attributes>
```

<value-constraint>

Defines the type of object to be allowed for the region entry values. This must be a fully-qualified class name. If value constraint isn't used, the region's value can be of any class. This attribute, along with `key-constraint`, is useful for querying and indexing because it provides object type information to the query engine.

Note: Set the constraint in every cache where you create or update the region entries. For client/server installations, match constraints between client and server and between clusters. The constraint is only checked in the cache that does the entry `put` or `create` operation. To avoid deserializing the object, the constraint is not checked when the entry is distributed to other caches.

Default: not set

API: `org.apache.geode.cache.RegionFactory.setValueConstraint`

Example:

```
<region-attributes>
  <value-constraint>
   cacheRunner.Portfolio
  </value-constraint>
</region-attributes>
```

<region-time-to-live>

Expiration setting that specifies how long the region can remain in the cache without anyone accessing or updating it.

Default: not set - no expiration of this type

API: `org.apache.geode.cache.RegionFactory.setRegionTimeToLive`

Example:

```
<region-attributes
  statistics-enabled="true">
  <region-time-to-live>
    <expiration-attributes
      timeout="3600"
      action="local-destroy"/>
  </region-time-to-live>
</region-attributes>
```

<expiration-attributes>

Within the `entry-time-to-live` or `entry-idle-time` element, this element specifies the expiration rules for removing old region entries that you are not using. You can destroy or invalidate entries, either locally or across the cluster. Within the `region-time-to-live` or `region-idle-time` element, this element specifies the expiration rules for the entire region.

API: See APIs for `entry-time-to-live`, `entry-idle-time`, `region-time-to-live`, `region-idle-time`

<expiration-attributes> Attributes

Attribute	Description	Default
timeout	Number of seconds before a region or an entry expires. If timeout is not specified, it defaults to zero (which means no expiration).	0
action	Action that should take place when a region or an entry expires. Select one of the following expiration actions: local-destroy: Removes the region or entry from the local cache, but does not distribute the removal operation to remote members. You cannot use this action on partitioned region entries. destroy: Removes the region or entry completely from the cache. Destroy actions are distributed according to the region's distribution settings. Use this option when the region or entry is no longer needed for any application in the cluster. invalidate: Default expiration action. Marks an entry or all entries in the region as invalid. Distributes the invalidation according to the region's scope. This is the proper choice when the region or the entry is no longer valid for any application in the cluster. local-invalidate: Marks an entry or all entries in the region as invalid but does not distribute the operation. You cannot use this action on partitioned region entries. Local region invalidation is only supported for regions that are not configured as replicated regions.	invalidate

Example:

```
<region-attributes
  statistics-enabled="true">
   <entry-time-to-live>
      <expiration-attributes
        timeout="60"
        action="local-destroy"/>
   </entry-time-to-live>
</region-attributes>
```

<custom-expiry>

Specifies the custom class that implements `org.apache.geode.cache.CustomExpiry`. You define this class in order to override the region-wide settings for specific entries. See Configure Data Expiration for an example.

Specify the Java class and its initialization parameters with the `<class-name>` and `<parameter>` sub-elements.

API: `org.apache.geode.cache.RegionFactory.setCustomEntryIdleTimeout`, `setCustomeEntryTimeToLive`

Example:

```
<region-attributes>
    <expiration-attributes
       timeout="60"
       action="local-destroy">
      <custom-expiry>
        <class-name>
          com.megaconglomerate.mypackage.MyClass
        </class-name>
      </custom-expiry>
</region-attributes>
```

Specify the Java class and its initialization parameters with the `<class-name>` and `<parameter>` sub-elements.

<region-idle-time>

Expiration setting that specifies how long the region can remain in the cache without anyone accessing it.

Note: To ensure reliable read behavior across the partitioned region, use `region-time-to-live` for region expiration instead of this setting.

Default: not set - no expiration of this type

API: `org.apache.geode.cache.RegionFactory.setRegionIdleTimeout`

Example:

```
<region-attributes statistics-enabled="true">
  <region-idle-time>
    <expiration-attributes
      timeout="3600"
      action="local-destroy"/>
  </region-idle-time>
</region-attributes>
```

<expiration-attributes>

Within the `entry-time-to-live` or `entry-idle-time` element, this element specifies the expiration rules for removing old region entries that you are not using. You can destroy or invalidate entries, either locally or across the cluster. Within the `region-time-to-live` or `region-idle-time` element, this element specifies the expiration rules for the entire region.

API: See APIs for `entry-time-to-live`, `entry-idle-time`, `region-time-to-live`, `region-idle-time`

<expiration-attributes> Attributes

Attribute	Description	Default
timeout	Number of seconds before a region or an entry expires. If timeout is not specified, it defaults to zero (which means no expiration).	0
action	Action that should take place when a region or an entry expires. Select one of the following expiration actions:	invalidate

local-destroy	Removes the region or entry from the local cache, but does not distribute the removal operation to remote members. You cannot use this action on partitioned region entries.
destroy	Removes the region or entry completely from the cache. Destroy actions are distributed according to the region's distribution settings. Use this option when the region or entry is no longer needed for any application in the cluster.
invalidate	Default expiration action. Marks an entry or all entries in the region as invalid. Distributes the invalidation according to the region's scope. This is the proper choice when the region or the entry is no longer valid for any application in the cluster.
local-invalidate	Marks an entry or all entries in the region as invalid but does not distribute the operation. You cannot use this action on partitioned region entries. Local region invalidation is only supported for regions that are not configured as replicated regions.

Example:

```
<region-attributes
  statistics-enabled="true">
   <entry-time-to-live>
      <expiration-attributes
        timeout="60"
        action="local-destroy"/>
   </entry-time-to-live>
</region-attributes>
```

<custom-expiry>

Specifies the custom class that implements `org.apache.geode.cache.CustomExpiry`. You define this class in order to override the region-wide settings for specific entries. See Configure Data Expiration for an example.

Specify the Java class and its initialization parameters with the `<class-name>` and `<parameter>` sub-elements.

API: `org.apache.geode.cache.RegionFactory.setCustomEntryIdleTimeout`, `setCustomeEntryTimeToLive`

Example:

```
<region-attributes>
    <expiration-attributes
       timeout="60"
       action="local-destroy">
      <custom-expiry>
        <class-name>
          com.megaconglomerate.mypackage.MyClass
        </class-name>
      </custom-expiry>
</region-attributes>
```

Specify the Java class and its initialization parameters with the `<class-name>` and `<parameter>` sub-elements.

<entry-time-to-live>

Expiration setting that specifies how long the region's entries can remain in the cache without anyone accessing or updating them. See <expiration-attributes> for details.

Default: not set - no expiration of this type.

API: `org.apache.geode.cache.RegionFactory.setEntryTimeToLive`

Example:

```
<region-attributes
  statistics-enabled="true">
  <entry-time-to-live>
    <expiration-attributes
     timeout="60"
     action="local-destroy"/>
  </entry-time-to-live>
</region-attributes>
```

<expiration-attributes>

Within the `entry-time-to-live` or `entry-idle-time` element, this element specifies the expiration rules for removing old region entries that you are not using. You can destroy or invalidate entries, either locally or across the cluster. Within the `region-time-to-live` or `region-idle-time` element, this element specifies the expiration rules for the entire region.

API: See APIs for `entry-time-to-live`, `entry-idle-time`, `region-time-to-live`, `region-idle-time`

<expiration-attributes> Attributes

Attribute	Description	Default
timeout	Number of seconds before a region or an entry expires. If timeout is not specified, it defaults to zero (which means no expiration).	0

Attribute	Description	Default
action	Action that should take place when a region or an entry expires. Select one of the following expiration actions: **local-destroy**: Removes the region or entry from the local cache, but does not distribute the removal operation to remote members. You cannot use this action on partitioned region entries. **destroy**: Removes the region or entry completely from the cache. Destroy actions are distributed according to the region's distribution settings. Use this option when the region or entry is no longer needed for any application in the cluster. **invalidate**: Default expiration action. Marks an entry or all entries in the region as invalid. Distributes the invalidation according to the region's scope. This is the proper choice when the region or the entry is no longer valid for any application in the cluster. **local-invalidate**: Marks an entry or all entries in the region as invalid but does not distribute the operation. You cannot use this action on partitioned region entries. Local region invalidation is only supported for regions that are not configured as replicated regions.	invalidate

Example:

```
<region-attributes
  statistics-enabled="true">
   <entry-time-to-live>
      <expiration-attributes
        timeout="60"
        action="local-destroy"/>
   </entry-time-to-live>
</region-attributes>
```

<custom-expiry>

Specifies the custom class that implements `org.apache.geode.cache.CustomExpiry`. You define this class in order to override the region-wide settings for specific entries. See Configure Data Expiration for an example.

Specify the Java class and its initialization parameters with the `<class-name>` and `<parameter>` sub-elements.

API: `org.apache.geode.cache.RegionFactory.setCustomEntryIdleTimeout`, `setCustomeEntryTimeToLive`

Example:

```
<region-attributes>
    <expiration-attributes
       timeout="60"
       action="local-destroy">
      <custom-expiry>
        <class-name>
          com.megaconglomerate.mypackage.MyClass
        </class-name>
      </custom-expiry>
</region-attributes>
```

Specify the Java class and its initialization parameters with the `<class-name>` and `<parameter>` sub-elements.

<entry-idle-time>

Expiration setting that specifies how long the region's entries can remain in the cache without anyone accessing them. See <expiration-attributes> for details.

Note: To ensure reliable read behavior across the partitioned region, use `entry-time-to-live` for entry expiration instead of this setting.

API: `org.apache.geode.cache.RegionFactory.setEntryIdleTimeout`

Example:

```
<region-attributes
  statistics-enabled="true">
  <entry-idle-time>
    <expiration-attributes
      timeout="60"
      action="local-invalidate"/>
    </expiration-attributes>
  </entry-idle-time>
</region-attributes>
```

<expiration-attributes>

Within the `entry-time-to-live` or `entry-idle-time` element, this element specifies the expiration rules for removing old region entries that you are not using. You can destroy or invalidate entries, either locally or across the cluster. Within the `region-time-to-live` or `region-idle-time` element, this element specifies the expiration rules for the entire region.

API: See APIs for `entry-time-to-live`, `entry-idle-time`, `region-time-to-live`, `region-idle-time`

<expiration-attributes> Attributes

Attribute	Description	Default
timeout	Number of seconds before a region or an entry expires. If timeout is not specified, it defaults to zero (which means no expiration).	0

Attribute	Description	Default
action	Action that should take place when a region or an entry expires. Select one of the following expiration actions: local-destroy: Removes the region or entry from the local cache, but does not distribute the removal operation to remote members. You cannot use this action on partitioned region entries. destroy: Removes the region or entry completely from the cache. Destroy actions are distributed according to the region's distribution settings. Use this option when the region or entry is no longer needed for any application in the cluster. invalidate: Default expiration action. Marks an entry or all entries in the region as invalid. Distributes the invalidation according to the region's scope. This is the proper choice when the region or the entry is no longer valid for any application in the cluster. local-invalidate: Marks an entry or all entries in the region as invalid but does not distribute the operation. You cannot use this action on partitioned region entries. Local region invalidation is only supported for regions that are not configured as replicated regions.	invalidate

Example:

```
<region-attributes
  statistics-enabled="true">
   <entry-time-to-live>
      <expiration-attributes
        timeout="60"
        action="local-destroy"/>
   </entry-time-to-live>
</region-attributes>
```

<custom-expiry>

Specifies the custom class that implements `org.apache.geode.cache.CustomExpiry`. You define this class in order to override the region-wide settings for specific entries. See Configure Data Expiration for an example.

Specify the Java class and its initialization parameters with the `<class-name>` and `<parameter>` sub-elements.

API: `org.apache.geode.cache.RegionFactory.setCustomEntryIdleTimeout`, `setCustomeEntryTimeToLive`

Example:

```
<region-attributes>
    <expiration-attributes
       timeout="60"
       action="local-destroy">
      <custom-expiry>
        <class-name>
          com.megaconglomerate.mypackage.MyClass
        </class-name>
      </custom-expiry>
</region-attributes>
```

Specify the Java class and its initialization parameters with the `<class-name>` and `<parameter>` sub-elements.

<cache-loader>

An event-handler plug-in that allows you to program for cache misses. At most, one cache loader can be defined in each member for the region. For distributed regions, a cache loader may be invoked remotely from other members that have the region defined. When an entry get results in a cache miss in a region with a cache loader defined, the loader's load method is called. This method is usually programmed to retrieve data from an outside data source, but it can do anything required by your application.

For partitioned regions, if you want to have a cache loader, install an instance of the cache loader in every data store. Partitioned regions support partitioned loading, where each cache loader loads only the data entries in the local member. If data redundancy is configured, data is loaded only if the local member holds the primary copy.

API: `org.apache.geode.cache.RegionFactory.setCacheLoader`

Example:

```
<region-attributes>
  <cache-loader>
    <class-name>quickstart.SimpleCacheLoader</class-name>
  </cache-loader>
</region-attributes>
```

<cache-writer>

An event-handler plug-in that allows you to receive before-event notification for changes to the region and its entries. It also has the ability to cancel events. At most, one cache writer can be defined in each member for the region. A cache writer may be invoked remotely from other members that have the region defined.

API: `org.apache.geode.cache.RegionFactory.setCacheWriter`

Example:

```
<region-attributes>
  <cache-writer>
    <class-name>quickstart.SimpleCacheWriter</class-name>
  </cache-writer>
</region-attributes>
```

<cache-listener>

An event-handler plug-in that receives after-event notification of changes to the region and its entries. Any number of cache listeners can be defined for a region in any member. VMware GemFire offers several listener types with callbacks to handle data and process events. Depending on the `data-policy` and the `interest-policy` subscription attributes, a cache listener may receive only events that originate in the local cache, or it may receive those events along with events that originate remotely.

Specify the Java class for the cache listener and its initialization parameters with the `<class-name>` and `<parameter>` sub-elements. See <class-name> and <parameter>.

API: `org.apache.geode.cache.RegionFactory.addCacheListener`

Example:

```
<region-attributes>
  <cache-listener>
    <class-name>
      quickstart.SimpleCacheListener
    </class-name>
  </cache-listener>
</region-attributes>
```

<eviction-attributes>

Specifies whether and how to control a region's size. Size is controlled by removing least recently used (LRU) entries to make space for new ones. This may be done through destroy or overflow actions. You can configure your region for lru-heap-percentage with an eviction action of local-destroy using stored region attributes.

Default: Uses the lru-entry-count algorithm.

API: `org.apache.geode.cache.RegionFactory.setEvictionAttributes`

Example:

```
<region-attributes>
  <eviction-attributes>
    <lru-entry-count
      maximum="1000"
      action="overflow-to-disk"/>
  </eviction-attributes>
</region-attributes>
```

<lru-entry-count>

Using the maximum attribute, specifies maximum region capacity based on entry count.

<lru-entry-count> Attributes

Attribute	Description	Default
action	Set one of the following eviction actions: local-destroy: Entry is destroyed locally. Not available for replicated regions. overflow-to-disk: Entry is overflowed to disk and the value set to null in memory. For partitioned regions, this provides the most reliable read behavior across the region.	local-destroy
maximum	The maximum number of entries allowed in a region.	

<lru-heap-percentage>

Runs evictions when the VMware GemFire resource manager says to. The manager orders evictions when the total cache size is over the heap percentage limit specified in the manager configuration. You can declare a Java class that implements the ObjectSizer interface to measure the size of objects in the Region.

Specify the Java class and its initialization parameters with the `<class-name>` and `<parameter>` sub-elements. See <class-name> and <parameter>.

<lru-heap-percentage> Attributes

Attribute	Description	Default
action	Set one of the following eviction actions: local-destroy: Entry is destroyed locally. Not available for replicated regions. overflow-to-disk: Entry is overflowed to disk and the value set to null in memory. For partitioned regions, this provides the most reliable read behavior across the region.	local-destroy

<lru-memory-size>

Using the maximum attribute, specifies maximum region capacity based on the amount of memory used, in megabytes. You can declare a Java class that implements the ObjectSizer interface to measure the size of objects in the Region.

Specify the Java class and its initialization parameters with the `<class-name>` and `<parameter>` sub-elements. See <class-name> and <parameter>.

<lru-memory-size> Attributes

Attribute	Description	Default
action	Set one of the following eviction actions: local-destroy: Entry is destroyed locally. Not available for replicated regions. overflow-to-disk: Entry is overflowed to disk and the value set to null in memory. For partitioned regions, this provides the most reliable read behavior across the region.	local-destroy
maximum	The maximum amount of memory used in the region, in megabytes.	

<jndi-bindings>

Specifies the binding for a data-source used in transaction management. See Configuring Database Connections Using JNDI.

Example:

```
<jndi-bindings>
      <jndi-binding type="XAPooledDataSource"
    jndi-name="newDB2trans"
    init-pool-size="20"
    max-pool-size="100"
    idle-timeout-seconds="20"
    blocking-timeout-seconds="5"
    login-timeout-seconds="10"
    xa-datasource-class="org.apache.derby.jdbc.EmbeddedXADataSource"
    user-name="mitul"
    password="encrypted(83f0069202c571faf1ae6c42b4ad46030e4e31c17409e19a)">
         <config-property>
          <config-property-name>Description</config-property-name>
          <config-property-type>java.lang.String</config-property-type>
          <config-property-value>pooled_transact</config-property-value>
       </config-property>
          <config-property>
             <config-property-name>DatabaseName</config-property-name>
             <config-property-type>java.lang.String</config-property-type>
             <config-property-value>newDB</config-property-value>
          </config-property>
          <config-property>
             <config-property-name>CreateDatabase</config-property-name>
             <config-property-type>java.lang.String</config-property-type>
             <config-property-value>create</config-property-value>
          </config-property>
       . . .
      </jndi-binding>
   </jndi-bindings>
```

<jndi-binding>

For every datasource that is bound to the JNDI tree, there should be one `<jndi-binding>` element. This element describes the property and the configuration of the datasource. VMware GemFire uses the attributes of the `<jndi-binding>` element for configuration. Use the `<config-property>` element to configure properties for the datasource.

We recommend that you set the username and password with the `user-name` and `password` jndi-binding attributes rather than using the `<config-property>` element.

<jndi-binding> Attributes

Attribute	Description	Default
blocking-timeout-seconds	The number of seconds that a connection remains associated with a transaction. If this value is exceeded, the connection is disassociated from the transaction.	120
conn-pooled-datasource-class	Java class used for the `PooledDataSource` type.	
connection-url	URL for connecting to the datasource. **Note:** If you are connecting to a JCA data source driver that implements XA transactions (where the jndi-binding type is **XAPooledDataSource**), do not use this attribute. Instead, define configuration properties for your database. See <config-property> for an example.	
idle-timeout-seconds	The maximum number of seconds that a connection can remain idle in a pool. When this threshold is reached, the connection is removed.	600
init-pool-size	The initial pool size of a PooledConnection (an XAConnection or a non-XAConnection).	10
jdbc-driver-class	Java class used for the `SimpleDataSource` type.	
jndi-name	The `jndi-name` attribute is the key binding parameter. If the value of jndi-name is a DataSource, it is bound as java:/myDatabase, where myDatabase is the name you assign to your data source. If the data source cannot be bound to JNDI at runtime, VMware GemFire logs a warning.	

Attribute	Description	Default
login-timeout-seconds	The maximum number of seconds for which a thread seeking a connection from a connection pool may be blocked. If the thread is unable to obtain connection in the stipulated time, a PoolException is thrown If a connection is available the thread returns immediately. If an existing connection is not available and the maximum number of connections in the pool has not been reached, a new connection is created and the thread returns immediately with the connection. If a connection is not available, the thread blocks for the specified time while waiting for an available connection.	30
managed-conn-factory-class	If the Resource Adapter is of type ManagedDataSource, this class becomes the source of the PooledConnection. (This class interface complies with the J2CA Java 2 Connector Architecture.)	
max-pool-size	The maximum size of the PooledConnection.	30
password	Password to access the datasource.	
transaction-type	When the `type` attribute is set to `ManagedDataSource`, specifies the type of transaction. Set one of the following `transaction-type`s: XATransaction: Select this option when you want to use aManagedConnection interface with a Java Transaction Manager to define transaction boundaries. This option allows a ManagedDataSource to participate in a transaction with a VMware GemFire cache. NoTransaction: No transactional behavior is used. LocalTransaction: Select this option when using a ManagedDataSource that is not managed by the Java Transaction manager.	none

Attribute	Description		Default
type	Set one of the following types:		none
	XAPooledDataSource	Pooled SQL connections. For this type, you must also set the `xa-datasource-class` attribute.	
	ManagedDataSource	JNDI binding type for the J2EE Connector Architecture (JCA). ManagedConnectionFactory. For information about the ManagedConnection interface, see Interface ManagedConnection in the Oracle documentation.	
	PooledDataSource	Pooled SQL connections. For this type, you must also set the `conn-pooled-datasource-class` attribute.	
	SimpleDataSource	Single SQL connection. No pooling of SQL connections is done. Connections are generated on the fly and cannot be reused. For this type, you must also set the `jdbc-driver-class` attribute.	
user-name	User name to access to the datasource.		
xa-datasource-class	Java class used for the `XAPooledDataSource` type.		

<config-property>

A configuration property of the datasource. Use the sub-elements to identify the name, datatype, and value of the property.

Default:

API: ``

Example:

```
<config-property>
     <config-property-name>DatabaseName</config-property-name>
     <config-property-type>java.lang.String</config-property-type>
     <config-property-value>newDB</config-property-value>
</config-property>
```

Configuration properties vary depending on the database vendor. See Configuring Database Connections Using JNDI for examples of different configuration property configurations.

<config-property-name>

The name of this datasource property.

<config-property-type>

The data type of this datasource property.

<config-property-value>

The value of this datasource property.

<region>

Defines a region in the cache. See <region-attributes> for more details on configuring regions. You can specify zero or more subregions within a region. See Create and Access Data Subregions for restrictions on creating subregions. For example, you cannot create a partitioned subregion.

Default:

API: `org.apache.geode.cache.RegionFactory` or `org.apache.geode.cache.ClientRegionFactory`

<region> Attributes

Attribute	Description	Default
name	Specify the name for the region. See Region Management for details.	
refid	Used to apply predefined attributes to the region being defined. If the nested "region-attributes" element has its own "refid", then it will cause the "refid" on the region to be ignored. The "refid" region attribute can be set to the name of a RegionShortcut or a ClientRegionShortcut. For more information, see Region Shortcuts and Custom Named Region Attributes and Storing and Retrieving Region Shortcuts and Custom Named Region Attributes.	

Example:

```
<!--Using region shortcut-->
<region
  name="PartitionedRegion"
  refid="PARTITION_REDUNDANT">
...
</region>

<!-- Retrieving and storing attributes -->
<region-attributes
  id="myPartition"
  refid="PARTITION_REDUNDANT">
    <partition-attributes
   local-max-memory="512"/>
</region-attributes>

<!-- Attributes are retrieved and applied in the first region -->
<region name="PartitionedRegion1" refid="myPartition"/>
```

See<region-attributes> for a complete listing of region attributes.

<region-attributes>

Specifies a region attributes template that can be named (by `id`) and referenced (by `refid`) later in the `cache.xml` and through the API.

API: `org.apache.geode.cache.RegionFactory` or `org.apache.geode.cache.ClientRegionFactory`

<region-attributes> Attributes

Attribute	Description	Default

concurrency-level	Gives an estimate of the maximum number of application threads that will concurrently access a region entry at one time. This attribute does not apply to partitioned regions. This attribute helps VMware GemFire optimize the use of system resources and reduce thread contention. This sets an initial parameter on the underlying `java.util.ConcurrentHashMap` used for storing region entries.
	
	Note: Before you modify this, read the concurrency level description, then see the Java API documentation for `java.util.ConcurrentHashMap`.
	
	API: `setConcurrencyLevel`
	
	Example:
	
	`<region-attributes`
	`  concurrency-level="10">`
	`</region-attributes>`
		16 (threads)

Attribute	Description	Default
data-policy	Specifies how the local cache handles data for a region. This setting controls behavior such as local data storage and region initialization. **Note**: Configure the most common options using the region shortcuts, `RegionShortcut` and `ClientRegionShortcut`. The default `data-policy` of `normal` specifies local cache storage. The empty policy specifies no local storage. In the region shortcuts, empty corresponds to the settings with the string `PROXY`. You can use an empty region for event delivery to and from the local cache without the memory overhead of data storage. You can specify the following data policies: **empty**: No data storage in the local cache. The region always appears empty. Use this for event delivery to and from the local cache without the memory overhead of data storage - zero-footprint producers that only distribute data to others and zero-footprint consumers that only see events. To receive events with this, set the region's `subscription-attributes` `interest-policy` to all. **normal**: Data used locally (accessed with `get`s, stored with `put`s, etc.) is stored in the local cache. This policy allows the contents in the cache to differ from other caches. **partition**: Data is partitioned across local and remote caches using the automatic data distribution behavior of partitioned regions. Additional configuration is done in the `partition-attributes`. **replicate**: The region is initialized with the data from other caches. After initialization, all events for the distributed region are automatically copied into the local region, maintaining a replica of the entire distributed region in the local cache. Operations that would cause the contents to differ with other caches are not allowed. This is compatible with local `scope`, behaving the same as for normal. **persistent-partition**: Behaves the same as `partition` and also persists data to disk.	normal

Attribute	Description	Default

	persistent-replicate — Behaves the same as `replicate` and also persists data to disk.
	preloaded — Initializes like a replicated region, then, once initialized, behaves like a normal region.
	API: `setDataPolicy`
	Example:
	`<region-attributes data-policy="replicate"> </region-attributes>`
	This is similar to using a region shortcut with`refid`, however when you use the REPLICATE region shortcut, it automatically sets the region's scope to `distributed-ack`.
	`<region-attributes refid="REPLICATE"> </region-attributes>`
	If you use `data-policy`, you must set the scope explicitly.	

Nested values:

persistent-replicate	Behaves the same as `replicate` and also persists data to disk.
preloaded	Initializes like a replicated region, then, once initialized, behaves like a normal region.

API: `setDataPolicy`

Example:

```
<region-attributes
  data-policy="replicate">
</region-attributes>
```

This is similar to using a region shortcut with`refid`, however when you use the REPLICATE region shortcut, it automatically sets the region's scope to `distributed-ack`.

```
<region-attributes
  refid="REPLICATE">
</region-attributes>
```

If you use `data-policy`, you must set the scope explicitly.

Attribute	Description	Default

enable-async-conflation	For TCP/IP distributions between peers, specifies whether to allow aggregation of asynchronous messages sent by the producer member for the region. This is a special-purpose Boolean attribute that applies only when asynchronous queues are used for slow consumers. A false value disables conflation so that all asynchronous messages are sent individually. This special-purpose attribute gives you extra control over peer-to-peer communication between distributed regions using TCP/IP. This attribute does not apply to client/server communication or to communication using the UDP unicast or IP multicast protocols.
	
	 Note: To use this attribute, the `multicast-enabled` region attribute `disable-tcp` in `gemfire.properties` must be false (the default for both). In addition, asynchronous queues must be enabled for slow consumers, specified with the `async`* gemfire properties.
	
	 API: `setEnableAsyncConflation`
	
	 Example:
	
	 `<region-attributes`
	`  enable-async-conflation="false">`
	`</region-attributes>`
		true

enable-gateway	Determines whether the gateway is enabled for the region. When set to true, events in the region are sent to the defined gateway hubs. Used only with GemFire version 6.x gateway configurations. For GemFire 7.0 configuration, see the `gateway-sender-id` attribute of the `<region-attributes>` element.	false

enable-subscription-conflation	Boolean for server regions that specifies whether the server can conflate its messages to the client. A true value enables conflation.
	
	 Note: The client can override this setting with the `conflate-events` property in its `gemfire.properties`.
	
	 API: `setEnableSubscriptionConflation`
	
	 Example:
	
	 `<region-attributes`
	` enable-subscription-conflation="true">`
	` </region-attributes>`
		false

Attribute	Description	Default

gateway-sender-ids	Specifies one or more gateway sender IDs to use for distributing region events to remote VMware GemFire sites. Specify multiple IDs as a comma-separated list.
	
	API: `addGatewaySenderId`
	
	Example:
	
	`<region-attributes`
	`  gateway-sender-ids="nwsender,swsende`
	`r">`
	`</region-attributes>`
		not set

async-event-queue-ids	Specifies one or more asynchronous event queues to use for distributing region events an `AsyncEventListener` implementation (for example, for write-behind cache event handling). Specify multiple IDs as a comma-separated list.
	
	API: `addAsyncEventQueueId`
	
	Example:
	
	`<region-attributes`
	`  async-event-queue-ids="customerqueue,o`
	`rdersqueue">`
	`</region-attributes>`
		not set

hub-id	If the `enable-gateway` attribute is set to true, a comma-separated list of gateway hub IDs that receive events from the region. Used only with GemFire version 6.x gateway configurations. For GemFire 7.0 configuration, see the `gateway-sender-id` attribute of the `<region-attributes>` element.	null

id	Stores the region attribute settings in the cache with this identifier. Once stored, the attributes can be retrieved using the region attribute `refid`.
	
	API: `setId`
	
	Example:
	
	`<region-attributes`
	`  id="persistent-replicated">`
	`</region-attributes>`
		not set

Attribute	Description	Default

ignore-jta	Boolean that determines whether operations on this region participate in active JTA transactions or ignore them and operate outside of the transactions. This is primarily used in cache loaders, writers, and listeners that need to perform non-transactional operations on a region, such as caching a result set.
	
	API: `setIgnoreJTA`
	
	Example:
	
	`<region-attributes`
	`  ignore-jta="true">`
	`</region-attributes>`
		false

index-update-type	Specifies whether region indexes are maintained synchronously with region modifications, or asynchronously in a background thread. In the `cache.xml` file, this is set as a value, asynchronous or synchronous, assigned to the `index-update-type` region attribute. Set this through the API by passing a boolean to the `setIndexMaintenanceSynchronous` method.
	
	API: `setIndexMaintenanceSynchronous`
	
	Example:
	
	`<region-attributes`
	`  index-update-type="asynchronous">`
	`</region-attributes>`
		synchronous updates

initial-capacity	Together with the `load-factor` region attribute, sets the initial parameters on the underlying `java.util.ConcurrentHashMap` used for storing region entries.
	
	API: `setInitialCapacity`
	
	Example:
	
	`<region-attributes`
	`  initial-capacity="20">`
	`</region-attributes>`
		16

Attribute	Description	Default

is-lock-grantor	Determines whether this member defines itself as the lock grantor for the region at region creation time. This only specifies whether the member becomes lock grantor at creation and does not reflect the current state of the member's lock grantor status. The member's lock grantor status may change if another member subsequently defines the region with `is-lock-grantor` set to true. This attribute is only relevant for regions with `global` `scope`, as only they allow locking. It affects implicit and explicit locking.
	
	API: `setLockGrantor`
	
	Example:
	
	`<region-attributes`
	`  is-lock-grantor="true">`
	`</region-attributes>`
		false

load-factor	Together with the initial-capacity region attribute, sets the initial parameters on the underlying `java.util.ConcurrentHashMap` used for storing region entries. This must be a floating point number between 0 and 1, inclusive.
	
	Note: Before you set this attribute, read the discussion of initial capacity and load factor, then see the Java API documentation for `java.util.ConcurrentHashMap`.
	
	API: `setLoadFactor`
	
	Example:
	
	`<region-attributes`
	`  load-factor="0.85">`
	`</region-attributes>`
		.75

mirror-type	Deprecated	

multicast-enabled	Boolean that specifies whether distributed operations on a region should use multicasting. To enable this, multicast must be enabled for the cluster with the `mcast-port` `gemfire.properties` setting.
	
	API: `setMulticastEnabled`
	
	Example:
	
	`<region-attributes`
	`  multicast-enabled="true">`
	`</region-attributes>`
		false

Attribute	Description	Default

pool-name	Identifies the region as a client region and specifies the server pool the region is to use. The named pool must be defined in the client cache before the region is created. If this is not set, the region does not connect to the servers as a client region.
	
	API: `setPoolName`
	
	Examples:
	
	This declaration creates the region as a client region with a server pool named DatabasePool. This pool-name specification is required, as there are multiple pools in the client cache:
	
	`<client-cache>`
	`    <pool name="DatabasePool"`
	`   subscription-enabled="true">`
	`        ...`
	`    </pool>`
	`    <pool >`
	`   name="OtherPool"`
	`   subscription-enabled="true">`
	`        ...`
	`    </pool>`
	`    <region ...`
	`        <region-attributes`
	`          pool-name="DatabasePool">`
	`        </region-attributes>`
	`        ...`
	
	This declaration creates the region as a client region assigned the single pool that is defined for the client cache. Here the pool-name specification is implied to be the only pool that exists in the cache:
	
	`<client-cache>`
	`    <pool`
	`   name="publisher"`
	`   subscription-enabled="true">`
	`        ...`
	`    </pool>`
	`    <region`
	`   name="myRegion"`
	`   refid="CACHING_PROXY">`
	`    </region>`
	`</client-cache>`
		not set

Attribute	Description	Default

disk-store-name	Assigns the region to the disk store with this name from the disk stores defined for the cache. Persist region data to disk by defining the region as persistent using the Shortcut Attribute Options or data-policy settings. Overflow data to disk by implementing LRU eviction-attributes with an action of overflow to disk. Each disk store defines the file system directories to use, how data is written to disk, and other disk storage maintenance properties. In addition, the `disk-synchronous` region attribute specifies whether writes are done synchronously or asynchronously.
	
	API: `setDiskStoreName`
	
	Example:
	
	`<region-attributes`
	`  disk-store-name="myStoreA" >`
	`</region-attributes>`
		null

disk-synchronous	For regions that write to disk, boolean that specifies whether disk writes are done synchronously for the region.
	
	API: `setDiskSynchronous`
	
	Example:
	
	`<region-attributes`
	`  disk-store-name="myStoreA"`
	`  disk-synchronous="true">`
	`</region-attributes>`
		true

refid	Retrieves region shortcuts and user-defined named region attributes for attributes initialization
	
	API: `setRefId`
	
	Example:
	
	`<region-attributes`
	`  refid="persistent-replicated">`
	`  <!-- Override any stored`
	`    attribute settings that you`
	`    need to ... -->`
	`</region-attributes>`
		not set

Attribute	Description	Default
scope	Definition: Determines how updates to region entries are distributed to the other caches in the cluster where the region and entry are defined. Scope also determines whether to allow remote invocation of some of the region's event handlers, and whether to use region entry versions to provide consistent updates across replicated regions. **Note**: You can configure the most common of these options with region shortcuts in `RegionShortcut` and `ClientRegionShortcut`. **Note**: Server regions that are not partitioned must be replicated with `distributed-ack` or `global` scope. The region shortcuts that specify `REPLICATE` have `distributed-ack` scope. Set one of the following scope values: local: No distribution. The region is visible only to threads running inside the member. distributed-no-ack: Events are distributed to remote caches with no acknowledgement required. distributed-ack: Events are distributed to remote caches with receipt acknowledgement required. Region entry versions are used to provide consistent updates across members of the cluster. global: Events are distributed to remote caches with global locking to ensure distributed cache consistency. **API:** `setScope` **Example:** `<region-attributes scope="distributed-ack"> </region-attributes>`	distributed-no-ack

```
<region-attributes
  scope="distributed-ack">
</region-attributes>
```

Attribute	Description	Default

statistics-enabled	Boolean specifying whether to gather statistics on the region. Must be true to use expiration on the region. VMware GemFire provides a standard set of statistics for cached regions and region entries, which give you information for fine-tuning your cluster. Unlike other VMware GemFire statistics, statistics for local and distributed regions are not archived and cannot be charted. They are kept in instances of `org.apache.geode.cache.CacheStatistics` and made available through the region and its entries through the `Region.getStatistics` and `Region.Entry.getStatistics` methods.
	
	API: `setStatisticsEnabled`
	
	Example:
	
	`<region-attributes`
	`  statistics-enabled="true">`
	`</region-attributes>`
		false

cloning-enabled	Determines how `fromDelta` applies deltas to the local cache for delta propagation. When true, the updates are applied to a clone of the value and then the clone is saved to the cache. When false, the value is modified in place in the cache.
	
	API: `setCloningEnabled`
	
	Example:
	
	`<region-attributes`
	`  cloning-enabled="true">`
	`</region-attributes>`
		false

Attribute	Description	Default

concurrency-checks-enabled	Determines whether members perform checks to provide consistent handling for concurrent or out-of-order updates to distributed regions. See Consistency for Region Updates.
	
	Note: Applications that use a `client-cache` may want to disable concurrency checking in order to see all events for a region. VMware GemFire server members can continue using concurrency checks for the region, but they will pass all events to the client cache. This configuration ensures that the client sees all events, but it does not prevent the client cache from becoming out-of-sync with the server cache.
	
	API: `setConcurrencyChecksEnabled`
	
	Example:
	
	`<region-attributes`
	` concurrency-checks-enabled="true">`
	`</region-attributes>`
		true

off-heap	Specifies that the region uses off-heap memory to store entry values, including values for region entries and queue entries. The region will still use heap memory for everything else, such as entry keys and the ConcurrentHashMap.
	
	API: `setOffHeap`
	
	Example:
	
	`<region-attributes`
	`  off-heap="true">`
	`</region-attributes>`
		false

<index>

Describes an index to be created on a region. The index node, if any, should all come immediately after the “region-attributes” node. The “name” attribute is a required field which identifies the name of the index. See Working with Indexes for more information on indexes.

Default:

API: `org.apache.geode.cache.query.QueryService.createIndex, createKeyIndex, createHashIndex`

<index> Attributes

Attribute	Description	Default
name	Required. Name of the index.	
from-clause	Specifies the collection(s) of objects that the index ranges over. The from-clause must only contain one and only one region path.	
expression	Specifies the lookup value of the index.	
imports	String containing the imports used to create the index. String should be specified in the query language syntax with each import statement separated by a semicolon. The imports statement provides packages and classes used in variable typing in the indexed and FROM expressions.	
key-index	True or false. Whether the index should be a key index. If true, the region key specified in the indexed expression is used to evaluate queries	
type	Possible values are “hash” or “range”.	range

Example:

```
<region name=exampleRegion>
 <region-attributes . . . >
 </region-attributes>
 <index
  name="myIndex"
  from-clause="/exampleRegion"
  expression="status"/>
 <index
  name="myKeyIndex"
  from-clause="/exampleRegion"
  expression="id" key-index="true"/>
 <index
  name="myHashIndex"
  from-clause="/exampleRegion p"
  expression="p.mktValue" type="hash"/>
 ...
</region>
```

<entry>

An “entry” element describes an entry to be added to a region. Note that if an entry with the given key already exists in the region, it will be replaced.

Default:

API: `org.apache.geode.cache.Region.create`, `put`, `get`, `putAll`, `getAll`

Example:

```
<region ...>
 <region-attributes ...>
   ...
 </region-attributes>
 <entry>
   <key><string>MyKey</string></key>
   <value><string>MyValue</string></value>
 </entry>
</region>
```

<key>

Required. Describes the key in a region entry. A key can contain either a <string> or a <declarable> sub-element.

<string>

Specifies a String to be placed in a Region entry.

Example:

```
<region ...>
 <region-attributes ...>
   ...
 </region-attributes>
 <entry>
   <key><string>MyKey</string></key>
   <value><string>MyValue</string></value>
 </entry>
</region>
```

<declarable>

Specifies a Declarable object to be placed in a Region entry.

Specify the Java class and its initialization parameters with the `<class-name>` and `<parameter>` sub-elements.

API: `Declarable`

Example:

```
<parameter name="cacheserver">
  <declarable>
    <class-name>org.apache.geode.addon.cache.CacheServerInitializer</class-name>
    <parameter name="system.property.prefix">
    <string>cacheserver</string>
    </parameter>
  </declarable>
</parameter>
```

<value>

Required. Describes the value of a region entry. A `<value>` can contain either a `<string>` or a `<declarable>` sub-element.

<string>

Specifies a String to be placed in a Region entry.

Example:

```
<region ...>
 <region-attributes ...>
   ...
 </region-attributes>
 <entry>
   <key><string>MyKey</string></key>
   <value><string>MyValue</string></value>
 </entry>
</region>
```

<declarable>

Specifies a Declarable object to be placed in a Region entry.

Specify the Java class and its initialization parameters with the `<class-name>` and `<parameter>` sub-elements.

API: `Declarable`

Example:

```
<parameter name="cacheserver">
  <declarable>
    <class-name>org.apache.geode.addon.cache.CacheServerInitializer</class-name>
    <parameter name="system.property.prefix">
    <string>cacheserver</string>
    </parameter>
  </declarable>
</parameter>
```

<region>

Defines a region in the cache. See <region-attributes> for more details on configuring regions. You can specify zero or more subregions within a region. See Create and Access Data Subregions for restrictions on creating subregions. For example, you cannot create a partitioned subregion.

Default:

API: `org.apache.geode.cache.RegionFactory` or `org.apache.geode.cache.ClientRegionFactory`

<region> Attributes

Attribute	Description	Default
name	Specify the name for the region. See Region Management for details.	
refid	Used to apply predefined attributes to the region being defined. If the nested “region-attributes” element has its own “refid”, then it will cause the “refid” on the region to be ignored. The “refid” region attribute can be set to the name of a RegionShortcut or a ClientRegionShortcut. For more information, see Region Shortcuts and Custom Named Region Attributes and Storing and Retrieving Region Shortcuts and Custom Named Region Attributes.	

Example:

```
<!--Using region shortcut-->
<region
  name="PartitionedRegion"
  refid="PARTITION_REDUNDANT">
...
</region>

<!-- Retrieving and storing attributes -->
<region-attributes
  id="myPartition"
  refid="PARTITION_REDUNDANT">
    <partition-attributes
   local-max-memory="512"/>
</region-attributes>

<!-- Attributes are retrieved and applied in the first region -->
<region name="PartitionedRegion1" refid="myPartition"/>
```

See<region-attributes> for a complete listing of region attributes.

<function-service>

Configures the behavior of the function execution service.

Example:

```
<client-cache>
  ...
    </region>
  <function-service>
    <function>
      <class-name>com.myCompany.tradeService.cache.func.TradeCalc</class-name>
    </function>
  </function-service>
  ...
</client-cache>
```

<function>

Defines a function for registration in the function service

Specify the Java class for the function and its initialization parameters with the `<class-name>` and `<parameter>` sub-elements. See <class-name> and <parameter>.

Default:

API: `org.apache.geode.cache.execute.FunctionService`

Example:

```
<function>
  <class-name>
    com.myCompany.tradeService.cache.func.TradeCalc
  </class-name>
</function>
```

<resource-manager>

A memory monitor that tracks cache size as a percentage of total heap or off-heap memory and controls size by restricting access to the cache and prompting eviction of old entries from the cache. For tenured heap, used in conjunction with settings for JVM memory and Java garbage collection. For off-heap memory, used with the off-heap memory manager.

API: `org.apache.geode.cache.control.ResourceManager`

<resource-manager> Attributes

Attribute	Description	Default
critical-heap-percentage	Percentage of heap at or above which the cache is considered in danger of becoming inoperable due to garbage collection pauses or out of memory exceptions. Only one change to this attribute or critical heap percentage will be allowed at any given time and its effect will be fully realized before the next change is allowed. This feature requires additional VM flags to perform properly. See `setCriticalHeapPercentage()` for details.	0
eviction-heap-percentage	Set the percentage of heap at or above which the eviction should begin on Regions configured for HeapLRU eviction. Changing this value may cause eviction to begin immediately.	• 0, If no region is configured with heap eviction • If `critical-heap-percentage` is set to a non-zero value, 5% less than that value. • 80%, if `critical-heap-percentage` is not configured.
critical-off-heap-percentage	Percentage of off-heap memory at or above which the cache is considered in danger of becoming inoperable due to garbage collection pauses or out of memory exceptions.	0

Attribute	Description	Default
eviction-off-heap-percentage	Set the percentage of off-heap memory at or above which the eviction should begin on Regions configured for HeapLRU eviction.	• 0, If no region is configured with heap eviction • If `critical-off-heap-percentage` is set to a non-zero value, 5% less than that value. • 80%, if `critical-off-heap-percentage` is not configured.

Example:

```
<client-cache>
  ...
  <resource-manager
    critical-heap-percentage="99.9"
    eviction-heap-percentage="85"/>
  ...
</client-cache>
```

<serialization-registration>

Set of serializer or instantiator tags to register customer DataSerializer extensions or DataSerializable implementations respectively.

Example:

```
<serialization-registration>
  <instantiator id="30">
    <class-name>com.package.MyClass</class-name>
  </instantiator>
</serialization-registration>
```

<serializer>

Allows you to configure the DataSerializer for this VMware GemFire member. It registers a custom class which extends DataSerializer to support custom serialization of non-modifiable object types inside VMware GemFire.

Specify the Java class for the `DataSerializer` and its initialization parameters with the `<class-name>` sub-element.

API: You can also register a `DataSerializer` by using the `org.apache.geode.DataSerializer.register` API. Use the `org.apache.geode.Instantiator` API to register a `DataSerializable` implementation.

<instantiator>

An Instantiator registers a custom class which implements the `DataSerializable` interface to support custom object serialization inside VMware GemFire.

Specify the Java class and its initialization parameters with the `<class-name>` sub-element.

API: `DataSerializable`

You can also directly specify `<instantiator>` as a sub-element of `<client-cache>`. Use the `org.apache.geode.Instantiator` API to register a `DataSerializable` implementation as the serialization framework for the cache. The following table lists the attribute that can be specified for an `<instantiator>`.

<instantiator> Attributes

Attribute	Description	Default
id	Required. ID that the Instantiator should associate with the `DataSerializable` type.	

<initializer>

Used to specify a callback class (and optionally its parameters) that will be run after the cache is initialized. This element can be specified for both server and client caches.

Specify the Java class and its initialization parameters with the `<class-name>` and `<parameter>` sub-elements. See <class-name> and <parameter>.

Default:

API: `Declarable`

Example:

```
<initializer>
  <class-name>MyInitializer</class-name>
  <parameter name="members">
    <string>2</string>
  </parameter>
</initializer>
```

Region Shortcuts

This topic describes the various region shortcuts you can use to configure VMware GemFire regions.

Region shortcuts are groupings of pre-configured attributes that define the characteristics of a region. You can use a region shortcut as a starting point when configuring regions and you can add additional configurations to customize your application. To reference a region shortcut in a VMware GemFire `cache.xml` file, use the `refid` attribute of the `<region>` element. For example:

```
<region name="myRegion" refid="PARTITION_REDUNDANT"/>
```

You can override the default values and add additional configurations within a `<region-attributes>` element of the `cache.xml` file. For example, the following configuration overrides the local-max-memory setting and adds the recovery-delay attribute:

```
<region name="myRegion" refid="PARTITION_REDUNDANT">
    <region-attributes>
        <partition-attributes
          local-max-memory="512"
           recovery-delay=-1/>
    </region-attributes>
</region>
```

You can also create your own, named region shortcuts for common custom configurations. See Region Shortcuts and Custom Named Region Attributes.

To configure a region using the gfsh command-line tool, specify the shortcut name with the `--type` argument. For example:

```
gfsh>create region --name=myRegion --type=PARTITION_REDUNDANT
```

Note: If you change the cache.xml file that defines a region, you must restart the member before the changes take effect.

For more information about configuring regions, see Region Management.

For more information about using the various types of VMware GemFire regions and when to use them, see Region Types.

For a quick reference listing all region shortcuts, see Region Shortcuts Quick Reference.

Region Shortcuts Quick Reference

This section provides a quick reference for all region shortcuts.

Region Shortcuts Default Configurations summarizes the default configurations for each region shortcut. See the cross reference for additional information about each shortcut.

Table 1. Region Shortcuts Default Configurations

Region Shortcut	Region Attributes	Other Attributes
LOCAL	scope: local data-policy: NORMAL	
LOCAL_HEAP_LRU	scope: local data-policy: NORMAL	**Eviction Attributes** eviction-algorithm: lru-heap-percentage eviction-action: local-destroy

Region Shortcut	Region Attributes	Other Attributes
LOCAL_OVERFLOW	scope: local data-policy: NORMAL	**Eviction Attributes** eviction-algorithm: lru-heap-percentage eviction-action: overflow-to-disk
LOCAL_PERSISTENT	scope: local data-policy: PERSISTENT_REPLICATE	
LOCAL_PERSISTENT_OVERFLOW	scope: local data-policy: PERSISTENT_REPLICATE	**Eviction Attributes** eviction-algorithm: lru-heap-percentage eviction-action: overflow-to-disk
PARTITION	data-policy: PARTITION	
PARTITION_HEAP_LRU	data-policy: PARTITION	**Eviction Attributes** eviction-algorithm: lru-heap-percentage eviction-action: local-destroy
PARTITION_OVERFLOW	data-policy: PARTITION	**Eviction Attributes** eviction-algorithm: lru-heap-percentage eviction-action: overflow-to-disk
PARTITION_PERSISTENT	data-policy: PERSISTENT_PARTITION	
PARTITION_PERSISTENT_OVERFLOW	data-policy: PERSISTENT_PARTITION	**Eviction Attributes** eviction-algorithm: lru-heap-percentage eviction-action: overflow-to-disk
PARTITION_PROXY	data-policy: PARTITION	**Partition Attributes** local-max-memory: 0
PARTITION_PROXY_REDUNDANT	data-policy: PARTITION	**Partition Attributes** local-max-memory: 0 redundant-copies: 1
PARTITION_REDUNDANT	data-policy: PARTITION	**Partition Attributes** redundant-copies: 1

Region Shortcut	Region Attributes	Other Attributes
PARTITION_REDUNDANT_HEAP_LRU	data-policy: PARTITION	**Eviction Attributes** eviction-algorithm: lru-heap-percentage eviction-action: local-destroy **Partition Attributes** redundant-copies: 1
PARTITION_REDUNDANT_OVERFLOW	data-policy: PARTITION	**Eviction Attributes** eviction-algorithm: lru-heap-percentage eviction-action: overflow-to-disk **Partition Attributes** redundant-copies: 1
PARTITION_REDUNDANT_PERSISTENT	data-policy: PERSISTENT_PARTITION	**Partition Attributes** redundant-copies: 1
PARTITION_REDUNDANT_PERSISTENT_OVERFLOW	data-policy: PERSISTENT_PARTITION	**Eviction Attributes** eviction-algorithm: lru-heap-percentage eviction-action: overflow-to-disk **Partition Attributes** redundant-copies: 1
REPLICATE	data-policy: REPLICATE scope: distributed-ack	
REPLICATE_HEAP_LRU	data-policy: PRELOADED scope: distributed-ack	**Eviction Attributes** eviction-algorithm: lru-heap-percentage eviction-action: local-destroy *Note:* Normally, you cannot have a `local-destroy` eviction policy on replicated regions. However, the REPLICATE_HEAP_LRU region is not the same as other replicated regions. This region is preloaded and remains in synch by registering interest in the keys of its peer's replicated regions. **Subscription Attributes** interest-policy: all
REPLICATE_OVERFLOW	data-policy: REPLICATE scope: distributed-ack	**Eviction Attributes** eviction-algorithm: lru-heap-percentage eviction-action: overflow-to-disk

Region Shortcut	Region Attributes	Other Attributes
REPLICATE_PERSISTENT	data-policy: PERSISTENT_REPLICATE scope: distributed-ack	
REPLICATE_PERSISTENT_OVERFLOW	data-policy: PERSISTENT_REPLICATE scope: distributed-ack	**Eviction Attributes** eviction-algorithm: lru-heap-percentage eviction-action: overflow-to-disk
REPLICATE_PROXY	data-policy: EMPTY scope: distributed-ack	

Exceptions and System Failures

Your application needs to catch certain classes to handle all the exceptions and system failures thrown by VMware GemFire.

- `GemFireCheckedException`. This class is the abstract superclass of exceptions that are thrown and declared. Wherever possible, GemFire exceptions are checked exceptions. `GemFireCheckedException` is a VMware GemFire version of `java.lang.Exception`.
- `GemFireException`. This class is the abstract superclass of unchecked exceptions that are thrown to indicate conditions for which the developer should not normally need to check. You can look at the subclasses of `GemFireException` to see all the runtime exceptions in the GemFire system; see the class hierarchy in the online Java API documentation. `GemFireException` is a VMware GemFire version of java.lang.`RuntimeException`. You can also look at the method details in the `Region` API javadocs for VMware GemFire exceptions you may want to catch.
- `SystemFailure`. In addition to exception management, VMware GemFire provides a class to help you manage catastrophic failure in your cluster, particularly in your application. The Javadocs for this class provide extensive guidance for managing failures in your system and your application. See `SystemFailure` in the `org.apache.geode` package.

To see the exceptions thrown by a specific method, refer to the method's online Java documentation.

A VMware GemFire system member can also throw exceptions generated by third-party software such as JGroups or `java.lang` classes. For assistance in handling these exceptions, see the vendor documentation.

Memory Requirements for Cached Data

VMware GemFire solutions architects need to estimate resource requirements for meeting application performance, scalability and availability goals.

These requirements include estimates for the following resources:

- memory
- number of machines
- network bandwidth

The information here is only a guideline, and assumes a basic understanding of VMware GemFire. While no two applications or use cases are exactly alike, the information here should be a solid starting point, based on real-world experience. Much like with physical database design, ultimately the right configuration and physical topology for deployment is based on the performance requirements, application data access characteristics, and resource constraints (i.e., memory, CPU, and network bandwidth) of the operating environment.

Core Guidelines for VMware GemFire Data Region Design

The following guidelines apply to region design:

- For 32-bit JVMs: If you have a small data set (< 2GB) and a read-heavy requirement, you should be using replicated regions.
- For 64-bit JVMs: If you have a data set that is larger than 50-60% of the JVM heap space you can use replicated regions. For read heavy applications this can be a performance win. For write heavy applications you should use partitioned caches.
- If you have a large data set and you are concerned about scalability you should be using partitioned regions.
- If you have a large data set and can tolerate an on-disk subset of data, you should be using either replicated regions or partitioned regions with overflow to disk.
- If you have different data sets that meet the above conditions, then you might want to consider a hybrid solution mixing replicated and partition regions. Do not exceed 50 to 75% of the JVM heap size depending on how write intensive your application is.

Memory Usage Overview

The following guidelines should provide a rough estimate of the amount of memory consumed by your system.

Memory calculation about keys and entries (objects) and region overhead for them can be divided by the number of members of the cluster for data placed in partitioned regions only. For other regions, the calculation is for each member that hosts the region. Memory used by sockets, threads, and the small amount of application overhead for VMware GemFire is per member.

For each entry added to a region, the VMware GemFire cache API consumes a certain amount of memory to store and manage the data. This overhead is required even when an entry is overflowed or persisted to disk. Thus objects on disk take up some JVM memory, even when they are paged to disk. The Java cache overhead introduced by a region, using a 32-bit JVM, can be approximated as listed below.

Actual memory use varies based on a number of factors, including the JVM you are using and the platform you are running on. For 64-bit JVMs, the usage will usually be larger than with 32-bit JVMs. As much as 80% more memory may be required for 64-bit JVMs, due to object references and headers using more memory.

There are several additional considerations for calculating your memory requirements:

- **Size of your stored data.** To estimate the size of your stored data, determine first whether you are storing the data in serialized or non-serialized form. In general, the non-serialized form will be the larger of the two. See Determining Object Serialization Overhead

  Objects in VMware GemFire are serialized for storage into partitioned regions and for all distribution activities, including moving data to disk for overflow and persistence. For optimum performance, VMware GemFire tries to reduce the number of times an object is serialized and deserialized, so your objects may be stored in serialized or non-serialized form in the cache.

- **Application object overhead for your data.** When calculating application overhead, make sure to count the key as well as the value, and to count every object if the key and/or value is a composite object.

  The following section "Calculating Application Object Overhead" provides details on how to estimate the memory overhead of the keys and values stored in the cache.

Calculating Application Object Overhead

To compute the memory overhead of a Java object, perform the following steps:

1. **Determine the object header size.** Each Java object has an object header. For a 32-bit JVM, it is 8 bytes. For a 64-bit JVM with a heap less than or equal to 32GB, it is 12 bytes. For a 64-bit JVM with a heap greater than 32GB, it is 16 bytes.
2. **Determine the memory overhead of the fields of the object.** For every instance field (including fields from super classes), add in the field's size. For primitive fields the sizes are:
   - 8 for long and double
   - 4 for int and float
   - 2 for char and short
   - 1 for byte and boolean

   For object reference fields, the size is 8 bytes for 64-bit JVM with a heap greater than 32GB. For all other JVMs, use 4 bytes.
3. **Add up the numbers from Step 1 and 2 and round it up to the next multiple of 8.** The result is the memory overhead of that Java object.

Java arrays. To compute the memory overhead of a Java array, you would add the object header (since the array is an object) and a primitive int field that contains its size. Treat each element of the array as if it was an instance field. For example, a byte array of the size 100 bytes would have one object header, one int field, and 100 byte fields. Use the three step process described above to do the computation.

Serialized objects. When computing the memory overhead of a serialized value, remember that the serialized form is stored in a byte array. Therefore, to figure out how many bytes the serialized form contains, compute the memory overhead of a Java byte array of that size and then add in the size of the serialized value wrapper.

When a value is initially stored in the cache in serialized form, a wrapper around the value is introduced that is kept in memory for the life of that value even if the value is later deserialized. Although this wrapper is only used internally, it does add to the memory footprint. The wrapper is an object with one int field and one object reference.

If you are using partitioned regions, every value is initially stored in serialized form. For other region types only values that come from a remote member (peers or clients) are initially stored in serialized form. (This is the most common case.) However, if a local operation stores the value in the local JVM's cache, then the value will be stored in object form. A large number of operations can cause a value stored in serialized form to be deserialized. Any operation that needs the object form of the value to be local can cause this deserialization. If such operations are performed, then that value will be stored in object form (with the additional serialized wrapper) and the serialized form becomes garbage.

Note: An exception to this is if the serialized from is encoded with PDX, then setting `read-serialized` to true will keep the serialized form in the cache.

See Determining Object Serialization Overhead for additional information on how to calculate memory usage requirements for storing serialized objects.

Using Key Storage Optimization

Keys are stored in object form except for certain classes where the storage of keys is optimized. Key storage is optimized by replacing the entry's object reference to the key with one or two primitive fields on the entry that store the key's data "inline". The following rules apply to determine whether a key is stored "inline":

- If the key's class is `java.lang.Integer`, `java.lang.Long`, or `java.util.UUID`, then the key is always stored inline. The memory overhead for an inlined Integer or Long key is 0 (zero). The memory overhead for an inlined UUID is 8.
- If the key's class is `java.lang.String`, then the key will be inlined if the string's length is small enough.
    - For ASCII strings whose length is less than 8, the inline memory overhead is 0 (zero).
    - For ASCII strings whose length is less than 16, the inline memory overhead is 8.
    - For non-ASCII strings whose length is less then 4, the inline memory overhead is 0 (zero).
    - For non-ASCII strings whose length is less then 8 the inline memory overhead is 8.

  All other strings are not inlined.

When to disable inline key storage. In some cases, storing keys inline may introduce extra memory or CPU usage. If all of your keys are also referenced from some other object, then it is better to not inline the key. If you frequently ask for the key from the region, then you may want to keep the object form stored in the cache so that you do not need to recreate the object form constantly. Note that the basic operation of checking whether a key is in a region does not require the object form but uses the inline primitive data.

The key inlining feature can be disabled by specifying the following VMware GemFire property upon member startup:

```
-Dgemfire.DISABLE_INLINE_REGION_KEYS=true
```

Measuring Cache Overhead

This table gives estimates for the cache overhead in a 32-bit JVM. The overhead is required even when an entry is overflowed or persisted to disk. Actual memory use varies based on a number of factors, including the JVM type and the platform you run on. For 64-bit JVMs, the usage will usually be larger than with 32-bit JVMs and may be as much as 80% more.

When calculating cache overhead…	You should …
For each region **Note:** Memory consumption for object headers and object references can vary for 64-bit JVMs, different JVM implementations, and different JDK versions.	add 64 bytes per entry
And concurrency checking is disabled (it is enabled by default)	subtract 16 bytes per entry (See Overhead for Consistency Checks.)
And statistics are enabled for the region	add 16 bytes per entry
And the region is persisted	add 52 bytes per entry
And the region is overflow only	add 44 bytes per entry
And the region has an LRU eviction controller	add 16 bytes per entry
And the region has global scope	add 110 bytes per entry
And the region has entry expiration configured	add 112 bytes per entry
For each optional user attribute	add 40 bytes per entry plus the memory overhead of the user attribute object

For indexes used in querying, the overhead varies greatly depending on the type of data you are storing and the type of index you create. You can roughly estimate the overhead for some types of indexes as follows:

- If the index has a single value per region entry for the indexed expression, the index introduces at most 243 bytes per region entry. An example of this type of index is: `fromClause="/portfolios", indexedExpression="id"`. The maximum of 243 bytes per region entry is reached if each entry has a unique value for the indexed expression. The overhead is reduced if the entries do not have unique index values.
- If each region entry has more than one value for the indexed expression, but no two region entries have the same value for it, then the index introduces at most 236 C + 75 bytes per region entry, where C is the average number of values per region entry for the expression.

- Lucene indexes add approximately 737 bytes per entry. The other index overhead estimates listed here also apply to Lucene indexes.

Estimating Management and Monitoring Overhead

The VMware GemFire JMX management and monitoring system contributes to memory overhead and should be accounted for when establishing the memory requirements for your deployment. Specifically, the memory footprint of any processes (such as locators) that are running as JMX managers can increase.

For each resource in the cluster that is being managed and monitored by the JMX Manager (for example, each MXBean such as MemberMXBean, RegionMXBean, DiskStoreMXBean, LockServiceMXBean and so on), you should add 10 KB of required memory to the JMX Manager node.

Determining Object Serialization Overhead

VMware GemFire PDX serialization can provide significant space savings over Java Serializable in addition to better performance. In some cases we have seen savings of up to 65%, but the savings will vary depending on the domain objects. PDX serialization is most likely to provide the most space savings of all available options. DataSerializable is more compact, but it requires that objects are deserialized on access, so that should be taken into account. On the other hand, PDX serializable does not require deserialization for most operations, and because of that, it may provide greater space savings.

In any case, the kinds and volumes of operations that would be done on the server side should be considered in the context of data serialization, as VMware GemFire has to deserialize data for some types of operations (access). For example, if a function invokes a get operation on the server side, the value returned from the get operation will be deserialized in most cases (the only time it will not be deserialized is when PDX serialization is used and the read-serialized attribute is set). The only way to find out the actual overhead is by running tests, and examining the memory usage.

Some additional serialization guidelines and tips:

- If you are using compound objects, do not mix using standard Java serialization with with VMware GemFire serialization (either DataSerializable or PDX). Standard Java serialization functions correctly when mixed with VMware GemFire serialization, but it can end up producing many more serialized bytes.

  To determine if you are using standard Java serialization, specify the `-DDataSerializer.DUMP_SERIALIZED=true` upon process execution. Then check your log for messages of this form:

  ```
  DataSerializer Serializing an instance of <className>
  ```

  Any classes list are being serialized with standard Java serialization. You can optimize your serialization by handling those classes in a `PdxSerializer` or a `DataSerializer` or changing the class to be `PdxSerializable` or `DataSerializable`.

- A simple way to determine the serialized size of an object is to create an instance of that object and then call `DataSerializer.writeObject(obj dataOutput)` where “dataOutput” wraps a `ByteArrayOutputStream`. You can then ask the stream for its size, and it will return the serialized size. Make sure you have configured your `PdxSerializer` and/or `DataSerializer`(s) configured before you calling `writeObject`.

If you do want to estimate memory usage for PDX serialized data, the following table provides estimated sizes for various types when using PDX serialization:

Type	Memory Usage
boolean	1 byte
byte	1 byte
char	2 bytes
short	2 bytes
int	4 bytes
long	8 bytes
float	8 bytes
String	String.length + 3 bytes
Domain Object	9 bytes (for PDX header) + object serialization length (total all member fields) + 1 to 4 extra bytes (depends on the total size of Domain object)

A note of caution: If the domain object contains many domain objects as member fields, then the memory overhead of PDX serialization can be considerably more than other types of serialization.

Calculating Socket Memory Requirements

Servers always maintain two outgoing connections to each of their peers. So for each peer a server has, there are four total connections: two going out to the peer and two coming in from the peer.

The server threads that service client requests also communicate with peers to distribute events and forward client requests. If the server's VMware GemFire connection property *conserve-sockets* is set to true, these threads use the already-established peer connections for this communication.

If *conserve-sockets* is false (the default), each thread that services clients establishes two of its own individual connections to its server peers, one to send, and one to receive. Each socket uses a file descriptor, so the number of available sockets is governed by two operating system settings:

- maximum open files allowed on the system as a whole
- maximum open files allowed for each session

In servers with many threads servicing clients, if *conserve-sockets* is set to false, the demand for connections can easily overrun the number of available sockets. Even with *conserve-sockets* set to false, you can cap the number of these connections by setting the server's *max-threads* parameter.

Since each client connection takes one server socket on a thread to handle the connection, and since that server acts as a proxy on partitioned regions to get results, or execute the function service on behalf of the client, for partitioned regions, if conserve-sockets is set to false, this also results in a new socket on the server being opened to each peer. Thus N sockets are opened, where N is the number of peers. Large number of clients simultaneously connecting to a large set of peers with a partitioned region with conserve sockets set to false can cause a large amount of memory to be consumed by sockets.

Note: There is also JVM overhead for the thread stack for each client connection being processed, set at 256KB or 512KB for most JVMs . On some JVMs you can reduce it to 128KB. You can use the VMware GemFire `max-threads` property or the VMware GemFire `max-connections` property to limit the number of client threads and thus both thread overhead and socket overhead.

The following table lists the memory requirements based on connections.

Connections	Memory requirements
Per socket	32,768 /socket (configurable) Default value per socket should be set to a number > 100 + sizeof (largest object in region) + sizeof (largest key)
If server (for example if there are clients that connect to it)	= (lesser of max-threads property on server or max-connections)* (socket buffer size +thread overhead for the JVM)
Per member of the cluster if conserve-sockets is set to true	4* number of peers
Per member, if conserve-sockets is set to false	4 * number of peers hosting that region* number of threads
If member hosts a Partitioned Region, If conserve-sockets set to false and it is a Server (this is cumulative with the above)	=< max-threads * 2 * number of peers **Note:** it is = 2* current number of clients connected * number of peers. Each connection spawns a thread.
Subscription Queues	
Per Server, depending on whether you limit the queue size. If you do, you can specify the number of megabytes or the number of entries until the queue overflows to disk. When possible, entries on the queue are references to minimize memory impact. The queue consumes memory not only for the key and the entry but also for the client ID/or thread ID as well as for the operation type. Since you can limit the queue to 1 MB, this number is completely configurable and thus there is no simple formula.	1 MB +
VMware GemFire classes and JVM overhead	Roughly 50MB
Thread overhead	
Each concurrent client connection into the a server results in a thread being spawned up to max-threads setting. After that a thread services multiple clients up to max-clients setting.	There is a thread stack overhead per connection (at a minimum 256KB to 512 KB, you can set it to smaller to 128KB on many JVMs.)

VMware GemFire Statistics List

This section describes the primary statistics gathered by VMware GemFire when statistics are enabled.

All statistics gathering requires the `gemfire.properties` `statistic-sampling-enabled` in `gemfire.properties` file to be true. Statistics that use time require the `gemfire.properties` `enable-time-statistics` to be true.

Performance statistics are collected for each Java application or cache server that connects to a cluster.

- Cache Performance (CachePerfStats)
- Cache Server (CacheServerStats)
- Client-Side Notifications (CacheClientUpdaterStats)
- Client-to-Server Messaging Performance (ClientStats)
- Client Connection Pool (PoolStats)
- Continuous Querying (CqQueryStats)
- Delta Propagation (DeltaPropagationStatistics)
- Disk Space Usage (DiskDirStatistics)
- Disk Store Statistics (DiskStoreStatistics)
- Disk Usage and Performance (DiskRegionStatistics)
- Distributed System Messaging (DistributionStats)
- Distributed Lock Services (DLockStats)
- Function Execution (FunctionServiceStatistics)
- Gateway Queue (GatewaySenderStatistics)
- Indexes (IndexStats)
- JVM Performance
- Locator (LocatorStats)
- Lucene Indexes (LuceneIndexStats)
- Off-Heap (OffHeapMemoryStats)
- Operating System Statistics - Linux
- Partitioned Regions (PartitionedRegion<partitioned_region_name>Statistics)
- Region Entry Eviction – Count-Based (LRUStatistics)
- Region Entry Eviction - Heap-based eviction (HeapLRUStatistics)
- Region Entry Eviction – Size-based (MemLRUStatistics)
- Server Notifications for All Clients (CacheClientNotifierStatistics)
- Server Notifications for Single Client (CacheClientProxyStatistics)
- Server-to-Client Messaging Performance (ClientSubscriptionStats)

- Statistics Collection (StatSampler)

Cache Performance (CachePerfStats)

Statistics for the VMware GemFire cache. These can be used to determine the type and number of cache operations being performed and how much time they consume.

Regarding VMware GemFire cache transactions, transaction-related statistics are compiled and stored as properties in the CachePerfStats statistic resource. Because the transaction's data scope is the cache, these statistics are collected on a per-cache basis.

The primary statistics are:

Statistic	Description
`cacheListenerCallsCompleted`	Total number of times a cache listener call has completed.
`cacheListenerCallsInProgress`	Current number of threads doing a cache listener call.
`cacheListenerCallTime`	Total time spent doing cache listener calls.
`cacheWriterCallsCompleted`	Total number of times a cache writer call has completed.
`cacheWriterCallsInProgress`	Current number of threads doing a cache writer call.
`cacheWriterCallTime`	Total time spent doing cache writer calls.
`compressions`	Total number of compression operations.
`compressTime`	Total time, in nanoseconds, spent compressing data.
`conflatedEvents`	The number of events that were conflated, and not delivered to event listeners or gateway senders on this member. Events are typically conflated because a later event was already applied to the cache, or because a concurrent event was ignored to ensure cache consistency. Note that some members may discard an update while other members apply the update, depending on the order in which each member receives the update. For this reason, the `conflatedEvents` statistic will differ for each VMware GemFire member. See Consistency for Region Updates.
`creates`	The total number of times an entry is added to this cache.
`decompressions`	Total number of decompression operations.
`decompressTime`	Total time, in nanoseconds, spent decompressing data.
`destroys`	The total number of times a cache object entry has been destroyed in this cache.

Statistic	Description
`diskTasksWaiting`	The current number of disk tasks, such as oplog compactions and asynchronous recoveries, that are waiting for a thread to run the operation.
`eventQueueSize`	The number of cache events waiting to be processed.
`eventQueueThrottleCount`	The total number of times a thread was delayed in adding an event to the event queue.
`eventQueueThrottleTime`	The total amount of time, in nanoseconds, spent delayed by the event queue throttle.
`eventThreads`	The number of threads currently processing events.
`getInitialImageKeysReceived`	Total number of keys received while doing getInitialImage operations.
`getInitialImagesCompleted`	Total number of times getInitialImages initiated by this cache have completed.
`getInitialImagesInProgressDesc`	Current number of getInitialImage operations currently in progress.
`getInitialImageTime`	Total time spent doing getInitialImages for region creation.
`getsDesc`	The total number of times a successful get has been done on this cache.
`getTime`	Total time spent doing get operations from this cache (including netsearch and netload).
`invalidates`	The total number of times an existing cache object entry value in this cache has been invalidated.
`loadsCompleted`	Total number of times a load on this cache has completed as a result of either a local get() or a remote netload.
`loadsInProgress`	Current number of threads in this cache doing a cache load.
`loadTime`	Total time spent invoking loaders on this cache.
`misses`	Total number of times a get on the cache did not find a value already in local memory. The number of hits (that is, gets that did not miss) can be calculated by subtracting misses from gets.
`netloadsCompleted`	Total number of times a network load initiated on this cache has completed.
`netloadsInProgress`	Current number of threads doing a network load initiated by a get() in this cache.
`netloadTime`	Total time spent doing network loads on this cache.

Statistic	Description
`netsearchesCompleted`	Total number of times network searches initiated by this cache have completed.
`netsearchesInProgress`	Current number of threads doing a network search initiated by a get() in this cache.
`netsearchTimeDesc`	Total time spent doing network searches for cache values.
`nonReplicatedTombstonesSize`	The approximate number of bytes that are currently consumed by tombstones in non-replicated regions. See Consistency for Region Updates.
`partitionedRegions`	The current number of partitioned regions in the cache.
`postCompressedBytes`	Total number of bytes after compressing.
`preCompressedBytes`	Total number of bytes before compressing.
`putAlls`	The total number of times a map is added or replaced in this cache as a result of a local operation. Note, this only counts putAlls done explicitly on this cache; it does not count updates pushed from other caches.
`putallTime`	Total time spent replacing a map in this cache as a result of a local operation. This includes synchronizing on the map, invoking cache callbacks, sending messages to other caches and waiting for responses (if required).
`puts`	The total number of times an entry is added or replaced in this cache as a result of a local operation (put(), create(), or get() which results in load, netsearch, or netloading a value). Note, this only counts puts done explicitly on this cache; it does not count updates pushed from other caches.
`putTime`	Total time spent adding or replacing an entry in this cache as a result of a local operation. This includes synchronizing on the map, invoking cache callbacks, sending messages to other caches, and waiting for responses (if required).
`queryExecutions`	Total number of times some query has been executed.
`queryExecutionTime`	Total time spent executing queries.
`regions`	The current number of regions in the cache.
`replicatedTombstonesSize`	The approximate number of bytes that are currently consumed by tombstones in replicated or partitioned regions. See Consistency for Region Updates.
`tombstoneCount`	The total number of tombstone entries created for performing concurrency checks. See Consistency for Region Updates.
`tombstoneGCCount`	The total number of tombstone garbage collection cycles that a member has performed. See Consistency for Region Updates.
`txCommitChanges`	Total number of changes made by committed transactions.

Statistic	Description
`txCommits`	Total number of times a transaction commit has succeeded.
`txCommitTime`	The total amount of time, in nanoseconds, spent doing successful transaction commits.
`txConflictCheckTime`	The total amount of time, in nanoseconds, spent doing conflict checks during transaction commit.
`txFailedLifeTime`	The total amount of time, in nanoseconds, spent in a transaction before a failed commit. The time measured starts at transaction begin and ends when commit is called.
`txFailureChanges`	Total number of changes lost by failed transactions.
`txFailures`	Total number of times a transaction commit has failed.
`txFailureTime`	The total amount of time, in nanoseconds, spent doing failed transaction commits.
`txRollbackChanges`	Total number of changes lost by explicit transaction rollbacks.
`txRollbackLifeTime`	The total amount of time, in nanoseconds, spent in a transaction before an explicit rollback. The time measured starts at transaction begin and ends when rollback is called.
`txRollbacks`	Total number of times a transaction has been explicitly rolled back.
`txRollbackTime`	The total amount of time, in nanoseconds, spent doing explicit transaction rollbacks.
`txSuccessLifeTime`	The total amount of time, in nanoseconds, spent in a transaction before a successful commit. The time measured starts at transaction begin and ends when commit is called.
`updates`	The total number of updates originating remotely that have been applied to this cache.
`updateTime`	Total time spent performing an update.

Cache Server (CacheServerStats)

Statistics used for cache servers and for gateway receivers are recorded in CacheServerStats in a cache server. The primary statistics are:

Statistic	Description
`abandonedReadRequests`	Number of read operations (requests) abandoned by clients.
`abandonedWriteRequests`	Number of write operations (requests) abandoned by clients.
`acceptsInProgress`	Current number of server accepts that are attempting to do the initial handshake with the client.
`acceptThreadStarts`	Total number of threads created (starts) to deal with an accepted socket. Note, this is not the current number of threads.
`batchSize`	The size (in bytes) of the batches received.
`clearRegionRequests`	Number of cache client operations clearRegion requests.

Statistic	Description
`clearRegionResponses`	Number of clearRegion responses written to the cache client.
`clientNotificationRequests`	Number of cache client operations notification requests.
`clientReadyRequests`	Number of cache client ready requests.
`clientReadyResponses`	Number of client ready responses written to the cache client.
`closeConnectionRequests`	Number of cache client close connection operations requests.
`connectionLoad`	The load from client to server connections as reported by the load probe installed in this server.
`connectionsTimedOut`	Total number of connections that have been timed out by the server because of client inactivity.
`connectionThreads`	Current number of threads dealing with a client connection.
`connectionThreadStarts`	Total number of threads created (starts) to deal with a client connection. Note, this is not the current number of threads.
`containsKeyRequests`	Number of cache client operations containsKey requests.
`containsKeyResponses`	Number of containsKey responses written to the cache client.
`currentClientConnections`	Number of sockets accepted.
`currentClients`	Number of client virtual machines (clients) connected.
`destroyRegionRequests`	Number of cache client operations destroyRegion requests.
`destroyRegionResponses`	Number of destroyRegion responses written to the cache client.
`destroyRequests`	Number of cache client operations destroy requests.
`destroyResponses`	Number of destroy responses written to the cache client.
`failedConnectionAttempts`	Number of failed connection attempts.
`getRequests`	Number of cache client operations get requests.
`getResponses`	Number of getResponses written to the cache client.
`loadPerConnection`	The estimate of how much load is added for each new connection as reported by the load probe installed in this server.
`loadPerQueue`	The estimate of how much load would be added for each new subscription connection as reported by the load probe installed in this server.
`messageBytesBeingReceived`	Current number of bytes consumed by messages being received or processed.
`messagesBeingReceived`	Current number of messages being received off the network or being processed after reception.
`outOfOrderGatewayBatchIds`	Number of Out of Order batch IDs (batches).
`processBatchRequests`	Number of cache client operations processBatch requests.
`processBatchResponses`	Number of processBatch responses written to the cache client.
`processBatchTime`	Total time, in nanoseconds, spent in processing a cache client processBatch request.

Statistic	Description
`processClearRegionTime`	Total time, in nanoseconds, spent in processing a cache client clearRegion request, including the time to clear the region from the cache.
`processClientNotificationTime`	Total time, in nanoseconds, spent in processing a cache client notification request.
`processClientReadyTime`	Total time, in nanoseconds, spent in processing a cache client ready request, including the time to destroy an object from the cache.
`processCloseConnectionTime`	Total time, in nanoseconds, spent in processing a cache client close connection request.
`processContainsKeyTime`	Total time spent, in nanoseconds, processing a containsKey request.
`processDestroyRegionTime`	Total time, in nanoseconds, spent in processing a cache client destroyRegion request, including the time to destroy the region from the cache.
`processDestroyTime`	Total time, in nanoseconds, spent in processing a cache client destroy request, including the time to destroy an object from the cache.
`processGetTime`	Total time, in nanoseconds, spent in processing a cache client get request, including the time to get an object from the cache.
`processPutAllTime`	Total time, in nanoseconds, spent in processing a cache client putAll request, including the time to put all objects into the cache.
`processPutTime`	Total time, in nanoseconds, spent in processing a cache client put request, including the time to put an object into the cache.
`processQueryTime`	Total time, in nanoseconds, spent in processing a cache client query request, including the time to destroy an object from the cache.
`processUpdateClientNotificationTime`	Total time, in nanoseconds, spent in processing a client notification update request.
`putAllRequests`	Number of cache client operations putAll requests.
`putAllResponses`	Number of putAllResponses written to the cache client.
`putRequests`	Number of cache client operations put requests.
`putResponses`	Number of putResponses written to the cache client.
`queryRequests`	Number of cache client operations query requests.
`queryResponses`	Number of query responses written to the cache client.
`queueLoad`	The load from subscription queues as reported by the load probe installed in this server
`readClearRegionRequestTime`	Total time, in nanoseconds, spent in reading clearRegion requests.
`readClientNotificationRequestTime`	Total time, in nanoseconds, spent in reading client notification requests.
`readClientReadyRequestTime`	Total time, in nanoseconds, spent in reading cache client ready requests.
`readCloseConnectionRequestTime`	Total time, in nanoseconds, spent in reading close connection requests.

Statistic	Description
`readContainsKeyRequestTime`	Total time, in nanoseconds, spent reading containsKey requests.
`readDestroyRegionRequestTime`	Total time, in nanoseconds, spent in reading destroyRegion requests.
`readDestroyRequestTime`	Total time, in nanoseconds, spent in reading destroy requests.
`readGetRequestTime`	Total time, in nanoseconds, spent in reading get requests.
`readProcessBatchRequestTime`	Total time, in nanoseconds, spent in reading processBatch requests.
`readPutAllRequestTime`	Total time, in nanoseconds, spent in reading putAll requests.
`readPutRequestTime`	Total time, in nanoseconds, spent in reading put requests.
`readQueryRequestTime`	Total time, in nanoseconds, spent in reading query requests.
`readUpdateClientNotificationRequestTime`	Total time, in nanoseconds, spent in reading client notification update requests.
`receivedBytes`	Total number of bytes received from clients.
`sentBytes`	Total number of bytes sent to clients.
`threadQueueSize`	Current number of connections waiting for a thread to start processing their message.
`updateClientNotificationRequests`	Number of cache client notification update requests.
`writeClearRegionResponseTime`	Total time, in nanoseconds, spent in writing clearRegion responses.
`writeClientReadyResponseTime`	Total time, in nanoseconds, spent in writing client ready responses.
`writeContainsKeyResponseTime`	Total time, in nanoseconds, spent writing containsKey responses.
`writeDestroyRegionResponseTime`	Total time, in nanoseconds, spent in writing destroyRegion responses.
`writeDestroyResponseTime`	Total time, in nanoseconds, spent in writing destroy responses.
`writeGetResponseTime`	Total time, in nanoseconds, spent in writing get responses.
`writeProcessBatchResponseTime`	Total time, in nanoseconds, spent in writing processBatch responses.
`writePutAllResponseTime`	Total time, in nanoseconds, spent in writing putAll responses.
`writePutResponseTime`	Total time, in nanoseconds, spent in writing put responses.
`writeQueryResponseTime`	Total time, in nanoseconds, spent in writing query responses.

Client-Side Notifications (CacheClientUpdaterStats)

Statistics in a client that pertain to server-to-client data pushed from the server over a queue to the client (they are the client side of the server's `CacheClientNotifierStatistics`) :

Statistic	Description
`receivedBytes`	Total number of bytes received from the server.
`messagesBeingReceived`	Current number of message being received off the network or being processed after reception.
`messageBytesBeingReceived`	Current number of bytes consumed by messages being received or processed.

Client-to-Server Messaging Performance (ClientStats & ClientSendStats)

These statistics are in a client and they describe all the messages sent from the client to a specific server. The primary statistics of ClientStats are:

Statistic	Description
`clearFailures`	Total number of clear attempts that have failed.
`clears`	Total number of clears completed successfully.
`clearsInProgress`	Current number of clears being executed.
`clearTime`	Total amount of time, in nanoseconds, spent doing clears.
`clearTimeouts`	Total number of clear attempts that have timed out.
`closeConFailures`	Total number of closeCon attempts that have failed.
`closeCons`	Total number of closeCons that have completed successfully.
`closeConsInProgress`	Current number of closeCons being executed.
`closeConTime`	Total amount of time, in nanoseconds, spent doing closeCons.
`closeConTimeouts`	Total number of closeCon attempts that have timed out.
`connections`	Current number of connections.
`connects`	Total number of times a connection has been created.
`containsKeyFailures`	Total number of containsKey attempts that have failed.
`containsKeys`	Total number of containsKeys that completed successfully.
`containsKeysInProgress`	Current number of containsKeys being executed.
`containsKeyTime`	Total amount of time, in nanoseconds, spent doing containsKeys.
`containsKeyTimeouts`	Total number of containsKey attempts that have timed out.
`destroyFailures`	Total number of destroy attempts that have failed.
`destroyRegionFailures`	Total number of destroyRegion attempts that have failed.
`destroyRegions`	Total number of destroyRegions that have completed successfully.
`destroyRegionsInProgress`	Current number of destroyRegions being executed.
`destroyRegionTime`	Total amount of time, in nanoseconds, spent doing destroyRegions.

Statistic	Description
`destroyRegionTimeouts`	Total number of destroyRegion attempts that have timed out.
`destroys`	Total number of destroys that have completed successfully.
`destroysInProgress`	Current number of destroys being executed.
`destroyTime`	Total amount of time, in nanoseconds, spent doing destroys.
`destroyTimeouts`	Total number of destroy attempts that have timed out.
`disconnects`	Total number of times a connection has been destroyed.
`gatewayBatchFailures`	Total number of gatewayBatch attempts that have failed.
`gatewayBatchs`	Total number of gatewayBatchs completed successfully.
`gatewayBatchsInProgress`	Current number of gatewayBatchs being executed.
`gatewayBatchTime`	Total amount of time, in nanoseconds, spent doing gatewayBatchs.
`gatewayBatchTimeouts`	Total number of gatewayBatch attempts that have timed out.
`getAllFailures`	Total number of getAll attempts that have failed.
`getAlls`	Total number of getAlls that have completed successfully.
`getAllsInProgress`	Current number of getAlls being executed.
`getAllTime`	Total amount of time, in nanoseconds, spent doing getAlls.
`getAllTimeouts`	Total number of getAll attempts that have timed out.
`getFailures`	Total number of get attempts that have failed.
`gets`	Total number of gets that have completed successfully.
`getsInProgress`	Current number of gets being executed.
`getTime`	Total amount of time, in nanoseconds, spent doing gets.
`getTimeouts`	Total number of get attempts that have timed out.
`keySetFailures`	Total number of keySet attempts that have failed.
`keySets`	Total number of keySets that have completed successfully.
`keySetsInProgress`	Current number of keySets being executed.
`keySetTime`	Total amount of time, in nanoseconds, spent doing keySets.
`keySetTimeouts`	Total number of keySet attempts that have timed out.
`makePrimaryFailures`	Total number of makePrimary attempts that have failed.
`makePrimarys`	Total number of makePrimarys that have completed successfully.
`makePrimarysInProgress`	Current number of makePrimarys being executed.
`makePrimaryTime`	Total amount of time, in nanoseconds, spent doing makePrimarys.
`makePrimaryTimeouts`	Total number of makePrimary attempts that have timed out.
`messageBytesBeingReceived`	Current number of bytes consumed by messages being received or processed.

Statistic	Description
`messagesBeingReceived`	Current number of messages being received off the network or being processed after reception.
`opFailures`	Total number of op attempts that have failed.
`ops`	Total number of ops that have completed successfully.
`opsInProgress`	Current number of ops being executed.
`opTime`	Total amount of time, in nanoseconds, spent doing ops.
`opTimeouts`	Total number of op attempts that have timed out.
`pingFailures`	Total number of ping attempts that have failed.
`pings`	Total number of pings that have completed successfully.
`pingsInProgress`	Current number of pings being executed.
`pingTime`	Total amount of time, in nanoseconds, spent doing pings.
`pingTimeouts`	Total number of ping attempts that have timed out.
`primaryAckFailures`	Total number of primaryAck attempts that have failed.
`primaryAcks`	Total number of primaryAcks that have completed successfully.
`primaryAckTime`	Total amount of time, in nanoseconds, spent doing primaryAcks.
`primaryAckTimeouts`	Total number of primaryAck attempts that have timed out.
`putAllFailures`	Total number of putAll attempts that have failed.
`putAlls`	Total number of putAlls that have completed successfully.
`putAllsInProgress`	Current number of putAlls being executed.
`putAllTime`	Total amount of time, in nanoseconds, spent doing putAlls.
`putAllTimeouts`	Total number of putAll attempts that have timed out.
`putFailures`	Total number of put attempts that have failed.
`puts`	Total number of puts that have completed successfully.
`putsInProgress`	Current number of puts being executed.
`putTime`	Total amount of time, in nanoseconds, spent doing puts.
`putTimeouts`	Total number of put attempts that have timed out.
`queryFailures`	Total number of query attempts that have failed.
`querys`	Total number of querys completed successfully.
`querysInProgress`	Current number of querys being executed.
`queryTime`	Total amount of time, in nanoseconds. spent doing querys.
`queryTimeouts`	Total number of query attempts that have timed out.
`readyForEvents`	Total number of readyForEventss that have completed successfully.

Statistic	Description
`readyForEventsFailures`	Total number of readyForEvents attempts that have failed.
`readyForEventsInProgress`	Current number of readyForEventss being executed
`readyForEventsTime`	Total amount of time, in nanoseconds, spent doing readyForEvents.
`readyForEventsTimeouts`	Total number of readyForEvents attempts that have timed out.
`receivedBytes`	Total number of bytes received from the server.
`registerInstantiators`	Total number of registerInstantiators completed successfully
`registerInstantiatorsFailures`	Total number of registerInstantiators attempts that have failed.
`registerInstantiatorssInProgress`	Current number of registerInstantiators being executed
`registerInstantiatorsTime`	Total amount of time, in nanoseconds, spent doing registerInstantiators.
`registerInstantiatorsTimeouts`	Total number of registerInstantiators attempts that have timed out.
`registerInterestFailures`	Total number of registerInterest attempts that have failed.
`registerInterests`	Total number of registerInterests that have completed successfully.
`registerInterestsInProgress`	Current number of registerInterests being executed.
`registerInterestTime`	Total amount of time, in nanoseconds, spent doing registerInterests.
`registerInterestTimeouts`	Total number of registerInterest attempts that have timed out.
`sentBytes`	Total number of bytes sent to the server.
`unregisterInterestFailures`	Total number of unregisterInterest attempts that have failed.
`unregisterInterests`	Total number of unregisterInterests that have completed successfully
`unregisterInterestsInProgress`	Current number of unregisterInterests being executed.
`unregisterInterestTime`	Total amount of time, in nanoseconds, spent doing unregisterInterests.
`unregisterInterestTimeouts`	Total number of unregisterInterest attempts that have timed out.

The primary statistics of ClientSendStats are:

Statistic	Description
`addPdxTypeSendFailures`	Total number of addPdxType operation's request messages not sent successfully from the client to server.
`addPdxTypeSendsSuccessful`	Total number of addPdxType operation's request messages sent successfully from the client to server.
`addPdxTypeSendsInProgress`	Current number of addPdxType operation's request messages being send from the client to server.
`addPdxTypeSendTime`	Total amount of time, in nanoseconds spent sending addPdxType operation's request messages successfully/unsuccessfully from the client to server.

Statistic	Description
`clearSendFailures`	Total number of clearSends that have failed.
`clearSends`	Total number of clearSends that have completed successfully.
`clearSendsInProgress`	Current number of clearSends being executed.
`clearSendTime`	Total amount of time, in nanoseconds, spent doing clearSends.
`closeConSendFailures`	Total number of closeConSends that have failed.
`closeConSends`	Total number of closeConSends that have completed successfully.
`closeConSendsInProgress`	Current number of closeConSends being executed.
`closeConSendTime`	Total amount of time, in nanoseconds, spent doing closeConSends.
`closeCQSendFailures`	Total number of closeCQ sends that have failed.
`closeCQSends`	Total number of closeCQ sends that have completed successfully.
`closeCQSendsInProgress`	Current number of closeCQ sends being executed.
`closeCQSendTime`	Total amount of time, in nanoseconds spent doing closeCQ sends.
`createCQSendFailures`	Total number of createCQ sends that have failed.
`createCQSends`	Total number of createCQ sends that have completed successfully.
`createCQSendsInProgress`	Current number of createCQ sends being executed.
`createCQSendTime`	Total amount of time, in nanoseconds spent doing createCQ sends.
`commitSendFailures`	Total number of commit sends that have failed.
`commitSends`	Total number of commit sends that have completed successfully.
`commitSendsInProgress`	Current number of commit sends being executed.
`commitSendTime`	Total amount of time, in nanoseconds spent doing commits.
`containsKeySendFailures`	Total number of containsKeySends that have failed.
`containsKeySends`	Total number of containsKeySends that have completed successfully.
`containsKeySendsInProgress`	Current number of containsKeySends being executed.
`containsKeySendTime`	Total amount of time, in nanoseconds, spent doing containsKeyends.
`destroyRegionSendFailures`	Total number of destroyRegionSends that have failed.
`destroyRegionSends`	Total number of destroyRegionSends that have completed successfully.
`destroyRegionSendsInProgress`	Current number of destroyRegionSends being executed.
`destroyRegionSendTime`	Total amount of time, in nanoseconds, spent doing destroyRegionSends.
`destroySendFailures`	Total number of destroySends that have failed.
`destroySends`	Total number of destroySends that have completed successfully.

Statistic	Description
`destroySendsInProgress`	Current number of destroySends being executed.
`destroySendTime`	Total amount of time, in nanoseconds, spent doing destroySends.
`executeFunctionSendFailures`	Total number of Function sends that have failed.
`executeFunctionSends`	Total number of Function sends that have completed successfully.
`executeFunctionSendsInProgress`	Current number of Function sends being executed.
`executeFunctionSendTime`	Total amount of time, in nanoseconds spent doing Function sends.
`gatewayBatchSendFailures`	Total number of gatewayBatchSends that have failed.
`gatewayBatchSends`	Total number of gatewayBatchSends that have completed successfully.
`gatewayBatchSendsInProgress`	Current number of gatewayBatchSends being executed.
`gatewayBatchSendTime`	Total amount of time, in nanoseconds, spent doing gatewayBatchSends.
`getAllSendFailures`	Total number of getAllSends that have failed.
`getAllSends`	Total number of getAllSends that have completed successfully.
`getAllSendsInProgress`	Current number of getAllSends being executed.
`getAllSendTime`	Total amount of time, in nanoseconds, spent doing getAllSends.
`getClientPartitionAttributesSendFailures`	Total number of getClientPartitionAttributes operation's request messages not sent successfully from the client to server.
`getClientPartitionAttributesSendsInProgress`	Current number of getClientPartitionAttributes operation's request messages being send from the client to server.
`getClientPartitionAttributesSendsSuccessful`	Total number of getClientPartitionAttributes operation's request messages sent successfully from the client to server.
`getClientPartitionAttributesSendTime`	Total amount of time, in nanoseconds spent sending getClientPartitionAttributes operation's request messages successfully/unsuccessfully from the client to server.
`getClientPRMetadataSendFailures`	Total number of getClientPRMetadata operation's request messages not sent successfully from the client to server.
`getClientPRMetadataSendsInProgress`	Current number of getClientPRMetadata operation's request messages being send from the client to server.
`getClientPRMetadataSendsSuccessful`	Total number of getClientPRMetadata operation's request messages sent successfully from the client to server.
`getClientPRMetadataSendTime`	Total amount of time, in nanoseconds spent sending getClientPRMetadata operation's request messages successfully/unsuccessfully from the client to server.
`getPDXIdForTypeSendFailures`	Total number of getPDXIdForType operation's request messages not sent successfully from the client to server.
`getPDXIdForTypeSendsInProgress`	Current number of getPDXIdForType operation's request messages being send from the client to server.

Statistic	Description
`getPDXIdForTypeSendsSuccessful`	Total number of getPDXIdForType operation's request messages sent successfully from the client to server.
`getPDXIdForTypeSendTime`	Total amount of time, in nanoseconds spent sending getPDXIdForType operation's request messages successfully/unsuccessfully from the client to server.
`getPDXTypeByIdSendFailures`	Total number of getPDXTypeById operation's request messages not sent successfully from the client to server.
`getPDXTypeByIdSendsInProgress`	Current number of getPDXTypeById operation's request messages being send from the client to server.
`getPDXTypeByIdSendsSuccessful`	Total number of getPDXTypeById operation's request messages sent successfully from the client to server.
`getPDXTypeByIdSendTime`	Total amount of time, in nanoseconds spent sending getPDXTypeById operation's request messages successfully/unsuccessfully from the client to server.
`getEntrySendFailures`	Total number of getEntry sends that have failed.
`getEntrySends`	Total number of getEntry sends that have completed successfully.
`getEntrySendsInProgress`	Current number of getEntry sends being executed.
`getEntrySendTime`	Total amount of time, in nanoseconds spent sending getEntry messages.
`getDurableCQsSendFailures`	Total number of getDurableCQs sends that have failed.
`getDurableCQsSends`	Total number of getDurableCQs sends that have completed successfully.
`getDurableCQsSendsInProgress`	Current number of getDurableCQs sends being executed.
`getDurableCQsSendTime`	Total amount of time, in nanoseconds spent doing getDurableCQs sends.
`getSendFailures`	Total number of getSends that have failed.
`getSends`	Total number of getSends that have completed successfully.
`getSendsInProgress`	Current number of getSends being executed.
`getSendTime`	Total amount of time, in nanoseconds, spent doing getSends.
`invalidateSendFailures`	Total number of invalidate sends that have failed.
`invalidateSends`	Total number of invalidate sends that have completed successfully.
`invalidateSendsInProgress`	Current number of invalidate sends being executed.
`invalidateSendTime`	Total amount of time, in nanoseconds spent doing invalidates.
`jtaSynchronizationSendFailures`	Total number of jtaSynchronization sends that have failed.
`jtaSynchronizationSends`	Total number of jtaSynchronization sends that have completed successfully.
`jtaSynchronizationSendsInProgress`	Current number of jtaSynchronization sends being executed.
`jtaSynchronizationSendTime`	Total amount of time, in nanoseconds spent doing jtaSynchronizations.

Statistic	Description
`keySetSendFailures`	Total number of keySetSends that have failed.
`keySetSends`	Total number of keySetSends that have completed successfully.
`keySetSendsInProgress`	Current number of keySetSends being executed.
`keySetSendTime`	Total amount of time, in nanoseconds, spent doing keySetSends.
`makePrimarySendFailures`	Total number of makePrimarySends that have failed.
`makePrimarySends`	Total number of makePrimarySends that have completed successfully.
`makePrimarySendsInProgress`	Current number of makePrimarySends being executed.
`makePrimarySendTime`	Total amount of time, in nanoseconds, spent doing makePrimarySends.
`pingSendFailures`	Total number of pingSends that have failed.
`pingSends`	Total number of pingSends that have completed successfully.
`pingSendsInProgress`	Current number of pingSends being executed.
`pingSendTime`	Total amount of time, in nanoseconds, spent doing pingSends.
`primaryAckSendFailures`	Total number of primaryAckSends that have failed.
`primaryAckSends`	Total number of primaryAckSends that have completed successfully.
`primaryAckSendTime`	Total amount of time, in nanoseconds, spent doing primaryAckSends.
`primaryAcksInProgress`	Current number of primaryAcks being executed.
`putAllSendFailures`	Total number of putAllSends that have failed.
`putAllSends`	Total number of putAllSends that have completed successfully.
`putAllSendsInProgress`	Current number of putAllSends being executed.
`putAllSendTime`	Total amount of time, in nanoseconds, spent doing putAllSends.
`putSendFailures`	Total number of putSends that have failed.
`putSends`	Total number of putSends that have completed successfully.
`putSendsInProgress`	Current number of putSends being executed.
`putSendTime`	Total amount of time, in nanoseconds, spent doing putSends.
`querySendFailures`	Total number of querySends that have failed.
`querySends`	Total number of querySends that have completed successfully.
`querySendsInProgress`	Current number of querySends being executed.
`querySendTime`	Total amount of time, in nanoseconds, spent doing querySends.
`readyForEventsSendFailures`	Total number of readyForEventsSends that have failed.
`readyForEventsSends`	Total number of readyForEventsSends that have completed successfully.

Statistic	Description
`readyForEventsSendsInProgress`	Current number of readyForEventsSends being executed.
`readyForEventsSendTime`	Total amount of time, in nanoseconds, spent doing readyForEventsSends.
`registerDataSerializersSendFailures`	Total number of registerDataSerializers sends that have failed.
`registerDataSerializersSends`	Total number of registerDataSerializers sends that have completed successfully.
`registerDataSerializersSendInProgress`	Current number of registerDataSerializers sends being executed.
`registerDataSerializersSendTime`	Total amount of time, in nanoseconds spent doing registerDataSerializers sends.
`registerInstantiatorsSendFailures`	Total number of registerInstantiators sends that have failed
`registerInstantiatorsSends`	Total number of registerInstantiators sends that have completed successfully
`registerInstantiatorsSendsInProgress`	Current number of registerInstantiators sends being executed
`registerInstantiatorsSendTime`	Total amount of time, in nanoseconds, spent doing registerInstantiatorsSends.
`registerInterestSendFailures`	Total number of registerInterestSends that have failed.
`registerInterestSends`	Total number of registerInterestSends that have completed successfully.
`registerInterestSendsInProgress`	Current number of registerInterestSends being executed.
`registerInterestSendTime`	Total amount of time, in nanoseconds, spent doing registerInterestSends.
`removeAllSendFailures`	Total number of removeAll sends that have failed.
`removeAllSends`	Total number of removeAll sends that have completed successfully.
`removeAllSendsInProgress`	Current number of removeAll sends being executed.
`removeAllSendTime`	Total amount of time, in nanoseconds spent doing removeAll sends.
`rollbackSendFailures`	Total number of rollback sends that have failed.
`rollbackSends`	Total number of rollback sends that have completed successfully.
`rollbackSendsInProgress`	Current number of rollback sends being executed.
`rollbackSendTime`	Total amount of time, in nanoseconds spent doing rollbacks.
`sizeSendFailures`	Total number of size sends that have failed.
`sizeSends`	Total number of size sends that have completed successfully.
`sizeSendsInProgress`	Current number of size sends being executed.

Statistic	Description
`sizeSendTime`	Total amount of time, in nanoseconds spent doing sizes.
`stopCQSendFailures`	Total number of stopCQ sends that have failed.
`stopCQSends`	Total number of stopCQ sends that have completed successfully.
`stopCQSendsInProgress`	Current number of stopCQ sends being executed.
`stopCQSendTime`	Total amount of time, in nanoseconds spent doing stopCQ sends.
`txFailoverSendFailures`	Total number of txFailover sends that have failed.
`txFailoverSends`	Total number of txFailover sends that have completed successfully.
`txFailoverSendsInProgress`	Current number of txFailover sends being executed.
`txFailoverSendTime`	Total amount of time, in nanoseconds spent doing txFailovers.
`unregisterInterestSendFailures`	Total number of unregisterInterestSends that have failed.
`unregisterInterestSends`	Total number of unregisterInterestSends that have completed successfully.
`unregisterInterestSendsInProgress`	Current number of unregisterInterestSends being executed.
`unregisterInterestSendTime`	Total amount of time, in nanoseconds, spent doing unregisterInterestSends.

Client Connection Pool (PoolStats)

These statistics are in a client and they describe one of the client's connection pools. The primary statistics are:

Statistic	Description
`connections`	Current number of connections.
`connectionWaits`	Total number of times a thread completed waiting for a connection (either by timing out or by getting a connection).
`connectionWaitsInProgress`	Current number of threads waiting for a connection.
`connectionWaitTime`	Total time, in nanoseconds, spent waiting for a connection.
`connects`	Total number of times a connection has been created.
`disconnects`	Total number of times a connection has been destroyed.
`ENDPOINTS_KNOWN`	Current number of servers discovered.
`idleChecks`	Total number of checks done for idle expiration.
`idleDisconnects`	Total number of disconnects done due to idle expiration.
`INITIAL_CONTACTS`	Number of contacts initially made the user.
`KNOWN_LOCATORS`	Current number of locators discovered.

Statistic	Description
`lifetimeChecks`	Total number of checks done for lifetime expiration.
`lifetimeConnects`	Total number of connects done due to lifetime expiration.
`lifetimeDisconnects`	Total number of disconnects done due to lifetime expiration.
`lifetimeExtensions`	Total number of times a connection's lifetime has been extended because the servers are still balanced.
`minPoolSizeConnects`	Total number of connects done to maintain minimum pool size.
`QUEUE_SERVERS`	Number of servers hosting this client.s subscription queue.
`REQUESTS_TO_LOCATOR`	Number of requests from this connection pool to a locator.
`RESPONSES_FROM_LOCATOR`	Number of responses from the locator to this connection pool.

Continuous Querying (CqQueryStats)

These statistics are for continuous querying information. The statistics are:

Statistic	Description
`CQS_CREATED`	Number of CQ operations created.
`CQS_ACTIVE`	Number of CQ operations actively executing.
`CQS_STOPPED`	Number of CQ operations stopped.
`CQS_CLOSED`	Number of CQ operations closed.
`CQS_ON_CLIENT`	Number of CQ operations on the client.
`CLIENTS_WITH_CQS`	Number of Clients with CQ operations.
`CQ_QUERY_EXECUTION_TIME`	Time taken, in nanoseconds, for CQ query execution.
`CQ_QUERY_EXECUTIONS_COMPLETED`	Number of CQ query executions operations.
`CQ_QUERY_EXECUTION_IN_PROGRESS`	CQ Query execution operations in progress.
`UNIQUE_CQ_QUERY`	Number of unique CQ queries.
`closeCQFailures`	Total number of closeCQ attempts that have failed. For client-to-server messaging performance.
`closeCQs`	Total number of closeCQs that have completed successfully. For client-to-server messaging performance.
`closeCQSendFailures`	Total number of closeCQSends that have failed. For client-to-server messaging performance.
`closeCQSends`	Total number of closeCQSends that have completed successfully. For client-to-server messaging performance.
`closeCQSendsInProgress`	Current number of closeCQSends being executed. For client-to-server messaging performance.

Statistic	Description
`closeCQSendTime`	Total amount of time, in nanoseconds, spent doing closeCQSends. For client-to-server messaging performance.
`closeCQsInProgress`	Current number of closeCQs being executed. For client-to-server messaging performance.
`closeCQTime`	Total amount of time, in nanoseconds, spent doing closeCQs. For client-to-server messaging performance.
`closeCQTimeouts`	Total number of closeCQ attempts that have timed out. For client-to-server messaging performance.
`createCQFailures`	Total number of createCQ attempts that have failed. For client-to-server messaging performance.
`createCQs`	Total number of createCQs that have completed successfully. For client-to-server messaging performance.
`createCQSendFailures`	Total number of createCQSends that have failed. For client-to-server messaging performance.
`createCQSends`	Total number of createCQSends that have completed successfully. For client-to-server messaging performance.
`createCQSendsInProgress`	Current number of createCQSends being executed. For client-to-server messaging performance.
`createCQSendTime`	Total amount of time, in nanoseconds, spent doing createCQSends. For client-to-server messaging performance.
`createCQsInProgress`	Current number of createCQs being executed. For client-to-server messaging performance.
`createCQTime`	Total amount of time, in nanoseconds, spent doing createCQs. For client-to-server messaging performance.
`createCQTimeouts`	Total number of createCQ attempts that have timed out. For client-to-server messaging performance.
`stopCQFailures`	Total number of stopCQ attempts that have failed. For client-to-server messaging performance.
`stopCQs`	Total number of stopCQs that have completed successfully. For client-to-server messaging performance.
`stopCQSendFailures`	Total number of stopCQSends that have failed. For client-to-server messaging performance.
`stopCQSends`	Total number of stopCQSends that have completed successfully. For client-to-server messaging performance.
`stopCQSendsInProgress`	Current number of stopCQSends being executed. For client-to-server messaging performance.
`stopCQSendTime`	Total amount of time, in nanoseconds, spent doing stopCQSends. For client-to-server messaging performance.
`stopCQsInProgress`	Current number of stopCQs being executed. For client-to-server messaging performance.

Statistic	Description
`stopCQTime`	Total amount of time, in nanoseconds, spent doing stopCQs. For client-to-server messaging performance.
`stopCQTimeouts`	Total number of stopCQ attempts that have timed out. For client-to-server messaging performance.
`cqCount`	Number of CQs operations on the client. For server notification to a single client.
`cqProcessingTime`	Total time, in nanoseconds, spent by the cache client notifier processing CQs. For server notification to all clients.

Delta Propagation (DeltaPropagationStatistics)

These statistics are for delta propagation between members. The primary statistics are:

Statistic	Description
`deltaFullValuePuts`	Total number of full value puts processed successfully in response to failed delta puts.
`deltaFullValueRequests`	Number of full value requests received from a client after failing to apply delta and processed successfully by this server.
`deltaMessageFailures`	The number of distribution messages containing delta that could not be processed at receiving side.
`deltaMessageFailures`	Current number of delta messages received but could not be processed after reception.
`deltaPutFailures`	Number of failures encountered while processing delta received from a client on this server.
`deltaPuts`	Total number of puts containing delta.
`deltaPutsTime`	Total amount of time, in nanoseconds, spent constructing delta part of puts.
`fullDeltaMessages`	Current number of full value delta messages received off network and processed after reception.
`fullDeltaRequests`	Number of full value requests made by this server to the sender client after failing to apply delta.
`fullValueDeltaMessagesRequested`	The number of distribution messages containing full value requested by this VMware GemFire system after failing to apply received delta.
`fullValueDeltaMessagesSent`	The number of distribution messages sent in response to full value requests by a remote VMware GemFire System as a result of failure in applying delta.
`partitionMessagesWithDeltaFailures`	Number of failures while processing PartitionMessages containing delta.
`partitionMessagesWithDeltaProcessed`	Number of PartitionMessages containing delta processed.
`partitionMessagesWithDeltaProcessedTime`	Total time spent applying deltas.
`partitionMessagesWithDeltaSent`	Number of PartitionMessages containing delta sent.

Statistic	Description
`partitionMessagesWithDeltaSentTime`	Total time spent extractng deltas.
`partitionMessagesWithFullValueDeltaRequested`	Number of requests for PartitionMessages containing full delta value as a result of failure in applying delta.
`partitionMessagesWithFullValueDeltaSent`	Number of PartitionMessages containing full delta value sent.
`preparedDeltaMessages`	The number of distribution messages containing delta that this VMware GemFire system has prepared for distribution.
`preparedDeltaMessages`	Number of client messages being prepared for dispatch, which have delta part in them.
`preparedDeltaMessagesTime`	The total amount of time this distribution manager has spent preparing delta parts of messages.
`processedDeltaMessages`	The number of distribution messages containing delta that this VMware GemFire system has processed.
`processedDeltaMessages`	Current number of delta messages received off network and processed after reception.
`processedDeltaMessagesTime`	The amount of time this distribution manager has spent in applying delta on its existing value.
`processedDeltaMessagesTime`	Total time spent applying received delta parts on existing messages at clients.
`processedDeltaPuts`	Number of cache client put requests containing delta received from a client and processed successfuly.
`processedDeltaPutsTime`	Total time spent in applying delta received from a client on existing value in this server's region.

Disk Space Usage (DiskDirStatistics)

These statistics pertain to the disk usage for a region's disk directory. The primary statistics are:

Statistic	Description
`diskSpace`	The total number of bytes currently being used on disk in this directory for oplog files.
`maximumSpace`	The configured maximum number of bytes allowed in this directory for oplog files. Note that some product configurations allow this maximum to be exceeded.
`volumeFreeSpace`	The total free space in bytes on the disk volume.
`volumeFreeSpaceChecks`	The total number of disk space checks.
`volumeFreeSpaceTime`	The total time, in nanseconds, spent checking disk usage.
`volumeSize`	The total size in bytes of the disk volume.

Disk Store Statistics (DiskStoreStatistics)

Statistics about a Region's use of the disk. The primary statistics are:

Statistic	Description
`backupsCompleted`	The number of backups of this disk store that have been taking while this VM was alive.
`backupsInProgress`	The current number of backups in progress on this disk store.
`compactableOplogs`	Current number of oplogs ready to be compacted.
`compactDeletes`	Total number of times an oplog compact did a delete.
`compactDeleteTime`	Total amount of time, in nanoseconds, spent doing deletes during a compact.
`compactInserts`	Total number of times an oplog compact did a db insert.
`compactInsertTime`	Total amount of time, in nanoseconds, spent doing inserts during a compact.
`compacts`	Total number of completed oplog compacts.
`compactsInProgress`	Current number of oplog compacts that are in progress.
`compactTime`	Total amount of time, in nanoseconds, spent compacting oplogs.
`compactUpdates`	Total number of times an oplog compact did an update.
`compactUpdateTime`	Total amount of time, in nanoseconds, spent doing updates during a compact.
`flushedBytes`	The total number of bytes written to disk by async queue flushes.
`flushes`	The total number of times the an entry has been flushed from the async queue.
`flushesInProgress`	Current number of oplog flushes that are in progress.
`flushTime`	The total amount of time spent doing an async queue flush.
`inactiveOplogs`	Current number of oplogs that are no longer being written but are not ready ready to compact.
`openOplogs`	Current number of oplogs this disk store has open.
`oplogReads`	Total number of oplog reads.
`oplogRecoveries`	The total number of oplogs recovered.
`oplogRecoveryTime`	The total amount of time spent doing an oplog recovery.
`oplogRecoveredBytes`	The total number of bytes that have been read from oplogs during a recovery.
`oplogSeeks`	Total number of oplog seeks.
`queueSize`	The current number of entries in the async queue waiting to be flushed to disk.
`readBytes`	The total number of bytes that have been read from disk.
`reads`	The total number of region entries that have been read from disk.
`readTime`	The total amount of time spent reading from disk.
`recoveriesInProgress`	Current number of persistent regions being recovered from disk.
`recoveredBytes`	The total number of bytes that have been read from disk during a recovery.

Statistic	Description
`recoveredEntryCreates`	The total number of entry create records processed while recovering oplog data.
`recoveredEntryDestroys`	The total number of entry destroy records processed while recovering oplog data.
`recoveredEntryUpdates`	The total number of entry update records processed while recovering oplog data.
`recoveredValuesSkippedDueToLRU`	The total number of entry values that did not need to be recovered due to the LRU.
`recoveryRecordsSkipped`	The total number of oplog records skipped during recovery.
`recoveryTime`	The total amount of time spent doing a recovery.
`removes`	The total number of region entries that have been removed from disk.
`removeTime`	The total amount of time spent removing from disk.
`uncreatedRecoveredRegions`	The current number of regions that have been recovered but have not yet been created.
`writes`	The total number of region entries that have been written to disk. A write is done every time an entry is created on disk or every time its value is modified on disk.
`writesInProgress`	Current number of oplog writes that are in progress.
`writeTime`	The total amount of time spent writing to disk.
`writtenBytes`	The total number of bytes that have been written to disk.

Disk Usage and Performance (DiskRegionStatistics)

Statistics regarding operations performed on a disk region for persistence/overflow. The primary statistics are:

Statistic	Description
`bufferSize`	Current number of bytes buffered to be written to the disk.
`bytesOnlyOnDisk`	The current number of bytes on disk and not in memory. It includes overflowed entries and recovered entries that have not yet been faulted in.
`commits`	Total number of commits.
`commitTime`	Total amount of time, in nanoseconds, spent doing commits.
`entriesInVM`	Current number of entries whose value resides in the member. The value may also have been written to the disk.
`entriesOnlyOnDisk`	Current number of entries whose value is on the disk and is not in memory. This is true of overflowed entries. It is also true of recovered entries that have not yet been faulted in.
`flushedBytes`	Total number of bytes flushed out of the async write buffer to the disk.
`flushes`	Total number of times the async write buffer has been flushed.

Statistic	Description
`flushTime`	Total amount of time, in nanoseconds, spent doing a buffer flush.
`readBytes`	Total number of bytes that have been read from the disk.
`reads`	Total number of region entries that have been read from the disk.
`readTime`	Total amount of time, in nanoseconds, spent reading from the disk.
`recoveredBytes`	Total number of bytes that have been read from disk during a recovery.
`recoveryTime`	Total amount of time, in nanoseconds, spent doing a recovery.
`removes`	Total number of region entries that have been removed from the disk.
`removeTime`	Total amount of time, in nanoseconds, spent removing from the disk.
`writes`	Total number of region entries that have been written to disk. A write is done every time an entry is created on disk or every time its value is modified on the disk.
`writeTime`	Total amount of time, in nanoseconds, spent writing to the disk.
`writtenBytes`	Total number of bytes that have been written to the disk.

Distributed System Messaging (DistributionStats)

Statistics on the VMware GemFire distribution layer. These statistics can be used to tell how much message traffic exists between this member and other cluster members.

The primary statistics are:

Statistic	Description
`asyncConflatedMsgsDesc`	The total number of queued conflated messages used for asynchronous queues.
`asyncDequeuedMsgsDesc`	The total number of queued messages that have been removed from the queue and successfully sent.
`asyncDistributionTimeoutExceededDesc`	Total number of times the async-distribution-timeout has been exceeded during a socket write.
`asyncQueueAddTime`	Total amount of time, in nanoseconds, spent in adding messages to async queue.
`asyncQueuedMsgsDesc`	The total number of queued messages used for asynchronous queues.
`asyncQueueFlushesCompletedDesc`	Total number of asynchronous queue flushes completed.
`asyncQueueFlushesInProgressDesc`	Current number of asynchronous queues being flushed.
`asyncQueueFlushTimeDesc`	Total time spent flushing asynchronous queues.
`asyncQueueRemoveTime`	Total amount of time, in nanoseconds, spent in removing messages from async queue.

Statistic	Description
`asyncQueuesDesc`	Current number of queues for asynchronous messaging.
`asyncQueueSizeDesc`	Current size in bytes used for asynchronous queues.
`asyncQueueSizeExceededDesc`	Total number of asynchronous queues that have exceeded the maximum size.
`asyncQueueTimeoutExceededDesc`	Total number of asynchronous queues that have timed out by being blocked for more than async-queue-timeout milliseconds.
`asyncSocketWriteBytes`	Total number of bytes sent out on non-blocking sockets.
`asyncSocketWriteRetries`	Total number of retries needed to write a single block of data using non-blocking socket write calls.
`asyncSocketWrites`	Total number of non-blocking socket write calls completed.
`asyncSocketWritesInProgress`	Current number of non-blocking socket write calls in progress.
`asyncSocketWriteTime`	Total amount of time, in nanoseconds, spent in non-blocking socket write calls.
`asyncThreadCompletedDesc`	Total number of iterations of work performed by asynchronous message queue threads.
`asyncThreadInProgressDesc`	Current iterations of work performed by asynchronous message queue threads.
`asyncThreadsDesc`	Total number of asynchronous message queue threads.
`asyncThreadTimeDesc`	Total time spent by asynchronous message queue threads performing iterations.
`batchSendTime`	Total amount of time, in nanoseconds, spent queueing and flushing message batches.
`batchWaitTime`	Reserved for future use
`broadcastMessagesDesc`	The number of distribution messages that the VMware GemFire system has broadcast. A broadcast message is one sent to every other manager in the group.
`broadcastMessagesTimeDesc`	The total amount of time this distribution manager has spent broadcasting messages. A broadcast message is one sent to every other manager in the group.
`bufferAcquires`	Total number of times a buffer has been acquired.
`bufferAcquiresInProgress`	Current number of threads waiting to acquire a buffer.
`bufferAcquireTime`	Total amount of time, in nanoseconds, spent acquiring a socket.
`commitWaitsDesc`	The number of transaction commits that had to wait for a response before they could complete.
`deserializations`	Total number of object deserialization calls.
`deserializationTime`	Total amount of time, in nanoseconds, spent deserializing objects.
`deserializedBytes`	Total number of bytes consumed by object deserialization.

Statistic	Description
`failedAcceptsDesc`	Total number of times an accept (receiver creation) of a connect from some other member has failed.
`failedConnectsDesc`	Total number of times a connect (sender creation) to some other member has failed.
`final`	String distributeMessageTimeDesc = The amount of time it takes to prepare a message and send it on the network. This includes sentMessagesTime.
`finalCheckRequestsReceived`	The number of final check requests that this member has received.
`finalCheckRequestsSent`	The number of final check requests that this member has sent.
`finalCheckResponsesReceived`	The number of final check responses that this member has received.
`finalCheckResponsesSent`	The number of final check responses that this member has sent.
`heartbeatRequestsSent`	The number of heartbeat request messages that this member has sent.
`heartbeatRequestsReceived`	The number of heartbeat request messages that this member has received.
`heartbeatsReceived`	The number of heartbeat messages that this member has received.
`heartbeatsSent`	The number of heartbeat messages that this member has sent.
`highPriorityQueueSizeDesc`	The number of high priority distribution messages currently waiting to be processed.
`highPriorityQueueThrottleCounDesc`	The total number of times a thread was delayed in adding a normal message to the high priority queue.
`highPriorityQueueThrottleTimeDesc`	The total amount of time, in nanoseconds, spent delayed by the high priority queue throttle.
`highPriorityThreadJobsDesc`	The number of messages currently being processed by high priority processor threads.
`highPriorityThreadsDesc`	The number of threads currently processing high priority messages.
`highPriorityThreadStarts`	Total number of times a thread has been created for the pool handling high priority messages.
`jgDirAckdownTime`	Time, in nanoseconds, spent in JGroups DirAck processing down events.
`jgDirAcksReceived`	Number of DirAck acks received.
`jgDirAckupTime`	Time, in nanoseconds, spent in JGroups DirAck processing up events.
`jgDISCOVERYdownTime`	Time, in nanoseconds, spent in JGroups DISCOVERY processing down events.
`jgDISCOVERYupTime`	Time, in nanoseconds, spent in JGroups DISCOVERY processing up events.
`jgFCdownTime`	Time, in nanoseconds, spent in JGroups FC processing down events.

Statistic	Description
`jgFCupTime`	Time, in nanoseconds, spent in JGroups FC processing up events.
`jgFDdownTime`	Time, in nanoseconds, spent in JGroups FD processing down events.
`jgFDupTime`	Time, in nanoseconds, spent in JGroups FD processing up events.
`jgFRAG2downTime`	Time, in nanoseconds, spent in JGroups FRAG2 processing down events.
`jgFRAG2upTime`	Time, in nanoseconds, spent in JGroups FRAG2 processing up events.
`jgFragmentationsPerformed`	Number of message fragmentation operations performed.
`jgFragmentsCreated`	Number of message fragments created.
`jgGMSdownTime`	Time, in nanoseconds, spent in JGroups GMS processing down events.
`jgGMSupTime`	Time, in nanoseconds, spent in JGroups GMS processing up events.
`jgNAKACKdownTime`	Time, in nanoseconds, spent in JGroups NAKACK processing down events.
`jgNAKACKupTime`	Time, in nanoseconds, spent in JGroups NAKACK processing up events.
`jgSTABLEdownTime`	Time, in nanoseconds, spent in JGroups STABLE processing down events.
`jgSTABLEupTime`	Time, in nanoseconds, spent in JGroups STABLE processing up events.
`jgTCPGOSSIPdownTime`	Time, in nanoseconds, spent in JGroups TCPGOSSIP processing down events.
`jgTCPGOSSIPupTime`	Time, in nanoseconds, spent in JGroups TCPGOSSIP processing up events.
`jgUDPdownTime`	Time, in nanoseconds, spent in JGroups UDP processing down events.
`jgUDPupTime`	Time, in nanosecnds, spent in JGroups UDP processing up events.
`jgUNICASTdownTime`	Time, in nanoseconds, spent in JGroups UNICAST processing down events.
`jgUNICASTupTime`	Time, in nanoseconds, spent in JGroups UNICAST processing up events.
`jgVIEWSYNCdownTime`	Time, in nanoseconds, spent in JGroups VIEWSYNC processing down events.
`jgVIEWSYNCupTime`	Time, in nanoseconds, spent in JGroups VIEWSYNC processing up events.
`lostConnectionLeaseDesc`	Total number of times an unshared sender socket has remained idle long enough that its lease expired.
`mcastReadBytes`	Total number of bytes received in multicast datagrams.
`mcastReads`	Total number of multicast datagrams received.
`mcastRetransmitRequests`	Total number of multicast datagram socket retransmission requests sent to other processes.
`mcastRetransmits`	Total number of multicast datagram socket retransmissions.
`mcastWriteBytes`	Total number of bytes sent out on multicast datagram sockets.
`mcastWrites`	Total number of multicast datagram socket write calls.
`mcastWriteTime`	Total amount of time, in nanoseconds, spent in multicast datagram socket write calls.

Statistic	Description
`messageBytesBeingReceived`	Current number of bytes consumed by messages being received or processed.
`messageChannelTimeDesc`	The total amount of time received messages spent in the distribution channel.
`messageProcessingScheduleTimeDesc`	The amount of time this distribution manager has spent dispatching a message to processor threads.
`messagesBeingReceived`	Current number of messages being received off the network or being processed after reception.
`msgDeserializationTime`	Total amount of time, in nanoseconds, spent deserializing messages.
`msgSerializationTime`	Total amount of time, in nanoseconds, spent serializing messages.
`nodesDesc`	The current number of members in this cluster.
`overflowQueueSizeDesc`	The number of normal distribution messages currently waiting to be processed.
`overflowQueueThrottleCountDesc`	The total number of times a thread was delayed in adding a normal message to the overflow queue.
`overflowQueueThrottleTimeDesc`	The total amount of time, in nanoseconds, spent delayed by the overflow queue throttle.
`partitionedRegionThreadJobsDesc`	The number of messages currently being processed by partitioned region threads.
`partitionedRegionThreadsDesc`	The number of threads currently processing partitioned region messages.
`partitionedRegionThreadStarts`	Total number of times a thread has been created for the pool handling partitioned region messages.
`pdxDeserializations`	Total number of PDX deserializations.
`pdxDeserializedBytes`	Total number of bytes read by PDX deserialization.
`pdxInstanceCreations`	Total number of times a PdxInstance has been created by deserialization.
`pdxInstanceDeserializations`	Total number of times getObject has been called on a PdxInstance.
`pdxInstanceDeserializationTime`	Total amount of time, in nanoseconds, spent deserializing PdxInstances by calling getObject.
`pdxSerializations`	Total number of PDX serializations.
`pdxSerializedBytes`	Total number of bytes produced by PDX serialization.
`processedMessagesDesc`	The number of distribution messages that the VMware GemFire system has processed.

Statistic	Description
processedMessagesTimeDesc	The amount of time this distribution manager has spent in message.process().
processingThreadJobsDesc	The number of messages currently being processed by pooled message processor threads.
processingThreadsDesc	The number of threads currently processing normal messages.
processingThreadStarts	Total number of times a thread has been created for the pool processing normal messages.
receivedBytesDesc	The number of distribution message bytes that the VMware GemFire system has received.
receivedMessagesDesc	The number of distribution messages that the VMware GemFire system has received.
receiverConnectionsDesc	Current number of sockets dedicated to receiving messages.
receiverDirectBufferSizeDesc	Current number of bytes allocated from direct memory as buffers for incoming messages.
receiverHeapBufferSizeDesc	Current number of bytes allocated from Java heap memory as buffers for incoming messages.S
reconnectAttemptsDesc	Total number of times an established connection was lost and a reconnect was attempted.
replyHandoffTimeDesc	Total number of seconds to switch thread contexts from processing thread to application thread.
replyMessageTimeDesc	The amount of time spent processing reply messages;
replyTimeoutsDesc	Total number of message replies that have timed out.
replyWaitMaxTimeDesc	Maximum time spent transmitting and then waiting for a reply to a message. See sentMessagesMaxTime for related information.
replyWaitsCompletedDesc	Total number of times waits for a reply have completed.
replyWaitsInProgressDesc	Current number of threads waiting for a reply.
replyWaitTimeDesc	Total time spent waiting for a reply to a message.
senderDirectBufferSizeDesc	Current number of bytes allocated from direct memory as buffers for outgoing messages.
senderHeapBufferSizeDesc	Current number of bytes allocated from Java heap memory as buffers for outgoing messages.
sentBytesDesc	The number of distribution message bytes that the VMware GemFire system has sent.
sentCommitMessagesDesc	The number of transaction commit messages that the VMware GemFire system has created to be sent. Note, it is possible for a commit to only create one message even though it will end up being sent to multiple recipients.

Statistic	Description
`sentMessagesDesc`	The number of distribution messages that the VMware GemFire system has sent, which includes broadcastMessages.
`sentMessagesMaxTimeDesc`	The highest amount of time this distribution manager has spent distributing a single message to the network.
`sentMessagesTimeDesc`	The total amount of time this distribution manager has spent sending messages, which includes broadcastMessagesTime.
`serializations`	Total number of object serialization calls.
`serializationTime`	Total amount of time, in nanoseconds, spent serializing objects.
`serializedBytes`	Total number of bytes produced by object serialization.
`serialPooledThreadDesc`	The number of threads created in the SerialQueuedExecutorPool.
`serialPooledThreadJobsDesc`	The number of messages currently being processed by pooled serial processor threads.
`serialPooledThreadStarts`	Total number of times a thread has been created for the serial pool(s).
`serialQueueBytesDesc`	The approximate number of bytes consumed by serial distribution messages currently waiting to be processed.
`serialQueueSizeDesc`	The number of serial distribution messages currently waiting to be processed.
`serialQueueThrottleCountDesc`	The total number of times a thread was delayed in adding a ordered message to the serial queue.
`serialQueueThrottleTimeDesc`	The total amount of time, in nanoseconds, spent delayed by the serial queue throttle.
`serialThreadJobsDesc`	The number of messages currently being processed by serial threads.
`serialThreadsDesc`	The number of threads currently processing serial/ordered messages.
`serialThreadStarts`	Total number of times a thread has been created for the serial message executor.
`sharedOrderedSenderConnectionsDesc`	Current number of shared sockets dedicated to sending ordered messages.
`sharedUnorderedSenderConnectionsDesc`	Current number of shared sockets dedicated to sending unordered messages.
`socketLocks`	Total number of times a socket has been locked.
`socketLockTime`	Total amount of time, in nanoseconds, spent locking a socket.
`suspectsReceived`	The number of suspect messages that this member has received.
`suspectsSent`	The number of suspect messages that this member has sent.
`syncSocketWriteBytes`	Total number of bytes sent out in synchronous/blocking mode on sockets.
`syncSocketWrites`	Total number of completed synchronous/blocking socket write calls.

Statistic	Description
`syncSocketWritesInProgress`	Current number of synchronous/blocking socket write calls in progress.
`syncSocketWriteTime`	Total amount of time, in nanoseconds, spent in synchronous/blocking socket write calls.
`tcpFinalCheckRequestsReceived`	The number of TCP final check requests that this member has received.
`tcpFinalCheckRequestsSent`	The number of TCP final check requests that this member has sent.
`tcpFinalCheckResponsesReceived`	The number of TCP final check responses that this member has received.
`tcpFinalCheckResponsesSent`	The number of TCP final check responses that this member has sent.
`threadOrderedSenderConnectionsDesc`	Current number of thread sockets dedicated to sending ordered messages.
`threadUnorderedSenderConnectionsDesc`	Current number of thread sockets dedicated to sending unordered messages.
`TOSentMsgs`	Total number of messages sent on thread owned senders.
`ucastReadBytes`	Total number of bytes received in unicast datagrams.
`ucastReads`	Total number of unicast datagrams received.
`ucastRetransmits`	Total number of unicast datagram socket retransmissions.
`ucastWriteBytes`	Total number of bytes sent out on unicast datagram sockets.
`ucastWrites`	Total number of unicast datagram socket write calls.
`ucastWriteTime`	Total amount of time, in nanoseconds, spent in unicast datagram socket write calls.
`udpDispatchRequestTime`	The total amount of time, in nanoseconds, spent deserializing and dispatching UDP messages in the message-reader thread.
`udpFinalCheckRequestsReceived`	The number of UDP final check requests that this member has received.
`udpFinalCheckRequestsSent`	The number of UDP final check requests that this member has sent.
`udpFinalCheckResponsesReceived`	The number of UDP final check responses that this member has received.
`udpFinalCheckResponsesSent`	The number of UDP final check responses that this member has sent.
`udpMsgDecryptionTime`	The total amount of time, in nanoseconds, spent decrypting UDP messages.
`udpMsgEncryptionTime`	The total amount of time, in nanoseconds, spent encrypting UDP messages.
`viewThreadJobsDesc`	The number of messages currently being processed by view threads.

Statistic	Description
`viewThreadsDesc`	The number of threads currently processing view messages.
`viewThreadStarts`	Total number of times a thread has been created for the view message executor.
`waitingQueueSizeDesc`	The number of distribution messages currently waiting for some other resource before they can be processed.
`waitingThreadJobsDesc`	The number of messages currently being processed by waiting pooly processor threads.
`waitingThreadsDesc`	The number of threads currently processing messages that had to wait for a resource.
`waitingThreadStarts`	Total number of times a thread has been created for the waiting pool.

Distribution Statistics Related to Slow Receivers

The distribution statistics provide statistics pertaining to slow receivers. The primary statistics are:

Statistic	Description
`asyncDistributionTimeoutExceeded`	Incremented every time an asyncSocketWrite has exceeded async-distribution-timeout and an async queue has been created.
`asyncQueue*`	Provide information about queues the producer is managing for its consumers. There are no statistics maintained for individual consumers. The following are the primary statistics of this type.
`asyncQueues`	Indicates the number of queues currently in the producer.
`asyncQueueSizeExceeded`	Incremented every time a queue has exceeded `async-max-queue-size` and the receiver has been sent a disconnect message.
`asyncQueueTimeoutExceeded`	Incremented every time a queue flushing has exceeded `async-queue-timeout` and the receiver has been sent a disconnect message.
`asyncSocketWrite*`	Used anytime a producer is distributing to one or more consumers with a non-zero distribution timeout. These statistics also reflect the writes done by the threads that service asynchronous queues.

Distributed Lock Services (DLockStats)

These statistics are for distributed lock services. The primary statistics are:

Statistic	Description
`becomeGrantorRequestsDesc`	Total number of times this member has explicitly requested to become lock grantor.
`createGrantorsCompletedDesc`	Total number of initial grantors created in this process.
`createGrantorsInProgressDesc`	Current number of initial grantors being created in this process.
`destroyReadsDesc`	The current number of DLockService destroy read locks held by this process.
`destroyReadWaitFailedTimeDesc`	Total time spent waiting for a DLockService destroy read lock that was not obtained.

Statistic	Description
`destroyReadWaitsCompletedDesc`	Total number of times a DLockService destroy read lock wait has completed successfully.
`destroyReadWaitsFailedDesc`	Total number of times a DLockService destroy read lock wait has completed unsuccessfully.
`destroyReadWaitsInProgressDesc`	Current number of threads waiting for a DLockService destroy read lock.
`destroyReadWaitTimeDesc`	Total time spent waiting for a DLockService destroy read lock that was obtained.
`destroyWritesDesc`	The current number of DLockService destroy write locks held by this process.
`destroyWriteWaitFailedTimeDesc`	Total time spent waiting for a DLockService destroy write lock that was not obtained.
`destroyWriteWaitsCompletedDesc`	Total number of times a DLockService destroy write lock wait has completed successfully.
`destroyWriteWaitsFailedDesc`	Total number of times a DLockService destroy write lock wait has completed unsuccessfully.
`destroyWriteWaitsInProgressDesc`	Current number of writes waiting for a DLockService destroy write lock.
`destroyWriteWaitTimeDesc`	Total time spent waiting for a DLockService destroy write lock that was obtained.
`grantorsDesc`	The current number of lock grantors hosted by this system member.
`grantorThreadExpireAndGrantLocksTimeDesc`	Total time spent by grantor thread(s) performing expireAndGrantLocks tasks.
`grantorThreadHandleRequestTimeoutsTimeDesc`	Total time spent by grantor thread(s) performing handleRequestTimeouts tasks.
`grantorThreadRemoveUnusedTokensTimeDesc`	Total time spent by grantor thread(s) performing removeUnusedTokens tasks.
`grantorThreadsCompletedDesc`	Total number of iterations of work performed by grantor thread(s).
`grantorThreadsInProgressDesc`	Current iterations of work performed by grantor thread.
`grantorThreadTimeDesc`	Total time spent by grantor thread(s) performing all grantor tasks.
`grantorWaitFailedTimeDesc`	Total time spent waiting for the grantor latch which resulted in failure.
`grantorWaitsCompletedDesc`	Total number of times waiting threads completed waiting for the grantor latch to open.
`grantorWaitsFailedDesc`	Total number of times waiting threads failed to finish waiting for the grantor latch to open.
`grantorWaitsInProgressDesc`	Current number of threads waiting for grantor latch to open.
`grantorWaitTimeDesc`	Total time spent waiting for the grantor latch which resulted in success.
`grantWaitDestroyedTimeDesc`	Total time spent granting of lock requests that failed because lock service was destroyed.
`grantWaitFailedTimeDesc`	Total time spent granting of lock requests that failed because try locks failed.

Statistic	Description
`grantWaitNotGrantorTimeDesc`	Total time spent granting of lock requests that failed because not grantor.
`grantWaitNotHolderTimeDesc`	Total time spent granting of lock requests that failed because reentrant was not holder.
`grantWaitsCompletedDesc`	Total number of times granting of a lock request has completed by successfully granting the lock.
`grantWaitsDestroyedDesc`	Total number of times granting of lock request failed because lock service was destroyed.
`grantWaitsFailedDesc`	Total number of times granting of lock request failed because try locks failed.
`grantWaitsInProgressDesc`	Current number of distributed lock requests being granted.
`grantWaitsNotGrantorDesc`	Total number of times granting of lock request failed because not grantor.
`grantWaitsNotHolderDesc`	Total number of times granting of lock request failed because reentrant was not holder.
`grantWaitsSuspendedDesc`	Total number of times granting of lock request failed because lock service was suspended.
`grantWaitsTimeoutDesc`	Total number of times granting of lock request failed because of a timeout.
`grantWaitSuspendedTimeDesc`	Total time spent granting of lock requests that failed because lock service was suspended.
`grantWaitTimeDesc`	Total time spent attempting to grant a distributed lock.
`grantWaitTimeoutTimeDesc`	Total time spent granting of lock requests that failed because of a timeout.
`lockReleasesCompletedDesc`	Total number of times distributed lock release has completed.
`lockReleasesInProgressDesc`	Current number of threads releasing a distributed lock.
`lockReleaseTimeDesc`	Total time spent releasing a distributed lock.
`lockWaitFailedTimeDesc`	Total number of times distributed lock wait has completed by failing to obtain the lock.
`lockWaitsCompletedDesc`	Total number of times distributed lock wait has completed by successfully obtaining the lock.
`lockWaitsFailedDesc`	Total time spent waiting for a distributed lock that failed to be obtained.
`lockWaitsInProgressDesc`	Current number of threads waiting for a distributed lock.
`lockWaitTimeDesc`	Total time spent waiting for a distributed lock that was obtained.
`pendingRequestsDesc`	The current number of pending lock requests queued by grantors in this process.
`requestQueuesDesc`	The current number of lock request queues used by this system member.
`serialQueueSizeDesc`	The number of serial distribution messages currently waiting to be processed.
`serialThreadsDesc`	The number of threads currently processing serial/ordered messages.
`serviceCreateLatchTimeDesc`	Total time spent creating lock services before releasing create latches.
`serviceCreatesCompletedDesc`	Total number of lock services created in this process.

Statistic	Description
`serviceCreatesInProgressDesc`	Current number of lock services being created in this process.
`serviceInitLatchTimeDesc`	Total time spent creating lock services before releasing init latches.
`servicesDesc`	The current number of lock grantors hosted by this system member.
`createGrantorTimeDesc`	Total time spent waiting create the initial grantor for lock services.
`tokensDesc`	The current number of lock tokens used by this system member.
`waitingQueueSizeDesc`	The number of distribution messages currently waiting for some other resource before they can be processed.
`waitingThreadsDesc`	The number of threads currently processing messages that had to wait for a resource.

Function Execution (FunctionStatistics)

These are the statistics for each execution of the function. The primary statistics are:

Statistic	Description
`functionExecutionCalls`	Total number of FunctionService.execute() calls for given function.
`functionExecutionsCompleted`	Total number of completed function.execute() calls for given function.
`functionExecutionsCompletedProcessingTime`	Total time consumed for all completed invocations of the given function.
`functionExecutionsExceptions`	Total number of Exceptions Occurred while executing function.
`functionExecutionsHasResultCompletedProcessingTime`	Total time consumed for all completed given function.execute() calls where hasResult() returns true.
`functionExecutionsHasResultRunning`	A gauge indicating the number of currently active execute() calls for functions where hasResult() returns true.
`functionExecutionsRunning`	number of currently running invocations of the given function.
`resultsReceived`	Total number of results received and passed to the ResultCollector.
`resultsSentToResultCollector`	Total number of results sent to the ResultCollector.

Gateway Queue (GatewaySenderStatistics)

These statistics are for outgoing gateway queue and its connection. The primary statistics are:

Statistic	Description
`batchDistributionTime`	Total time, in nanoseconds, spent distributing batches of events to other gateways.
`batchesDistributed`	Number of batches of events operations removed from the event queue and sent.
`batchesRedistributed`	Number of batches of events operations removed from the event queue and resent.
`batchesResized`	The number of batches resized due to a batch being too large.
`eventQueueSize`	Size of the event operations queue.

Statistic	Description
`eventQueueTime`	Total time, in nanoseconds, spent queueing events.
`eventsDistributed`	Number of events operations removed from the event queue and sent.
`eventsDroppedDueToPrimarySenderNotRunning`	Number of events dropped because the primary gateway sender is not running.
`eventsNotQueuedConflated`	Number of events operations received but not added to the event queue because the queue already contains an event with the event's key.
`eventsProcessedByPQRM`	Total number of events processed by the parallel queue removal message (PQRM).
`eventsQueued`	Number of events operations added to the event queue.
`secondaryEventQueueSize`	Size of the secondary event queue.
`unprocessedEventMapSize`	Current number of events entries in the secondary's unprocessed event map.
`unprocessedEventsAddedBySecondary`	Number of events added to the secondary's unprocessed event map by the secondary.
`unprocessedEventsRemovedByPrimary`	Number of events removed through a listener from the secondary's unprocessed event map by the primary.
`unprocessedEventsRemovedByTimeout`	Number of events removed from the secondary's unprocessed event map by a timeout.
`unprocessedTokenMapSize`	Current number of tokens entries in the secondary's unprocessed token map.
`unprocessedTokensAddedByPrimary`	Number of tokens added through a listener to the secondary's unprocessed token map by the primary.
`unprocessedTokensRemovedBySecondary`	Number of tokens removed from the secondary's unprocessed token map by the secondary.
`unprocessedTokensRemovedByTimeout`	Number of tokens removed from the secondary's unprocessed token map by a timeout.

Indexes (IndexStats)

Query-Independent Statistics on Indexes

Statistic	Description
`numKeys`	Number of keys currently stored in the Index.
`numUpdates`	Number of updates applied and completed on the Index while inserting, modifying , or deleting corresponding data in VMware GemFire.
`numValues`	Number of values currently stored in the Index.
`updatesInProgress`	Current number of updates in progress on the Index. Concurrent updates on an index are allowed.
`updateTime`	Total time taken in applying and completing updates on the Index.

Query-Dependent Statistics on Indexes

Statistic	Description
`numUses`	Number of times the Index has been used for querying.
`usesInProgress`	Current number of uses of the index in progress or current number of concurrent threads accessing the index for querying. Concurrent use of an index is allowed for different queries.
`useTime`	Total time during the use of the Index for querying.

JVM Performance

VMware GemFire JVM Resource Manager (ResourceManagerStats)

Statistics related to the VMware GemFire's resource manager. Use these to help analyze and tune your JVM memory settings and the VMware GemFire resource manager settings. The primary statistics are:

Statistic	Description
`criticalThreshold`	The cache resource-manager setting critical-heap-percentage.
`evictionStartEvents`	Number of times eviction activities were started due to the heap use going over the eviction threshold.
`evictionStopEvents`	Number of times eviction activities were stopped due to the heap use going below the eviction threshold.
`evictionThreshold`	The cache resource-manager setting eviction-heap-percentage.
`heapCriticalEvents`	Number of times incoming cache activities were blocked due to heap use going over the critical threshold.
`heapSafeEvents`	Number of times incoming cache activities were unblocked due to heap use going under the critical threshold.
`tenuredHeapUsed`	Percentage of tenured heap currently in use.

JVM Java Runtime (VMStats)

Show the JVM's Java usage and can be used to detect possible problems with memory consumption. These statistics are recorded from java.lang.Runtime under VMStats. The primary statistics are:

Statistic	Description
`cpus`	Number of CPUs available to the member on its machine.
`daemonThreads`	Current number of live daemon threads in this JVM.
`fdLimit`	Maximum number of file descriptors.
`fdsOpen`	Current number of open file descriptors.
`freeMemory`	An approximation for the total amount of memory, measured in bytes, currently available for future allocated objects.

Statistic	Description
`loadedClasses`	Total number of classes loaded since the JVM started.
`maxMemory`	The maximum amount of memory, measured in bytes, that the JVM will attempt to use.
`peakThreads`	High water mark of live threads in this JVM.
`pendingFinalization`	Number of objects that are pending finalization in the JVM.
`processCpuTime`	CPU time, measured in nanoseconds, used by the process.
`threads`	Current number of live threads (both daemon and non-daemon) in this JVM.
`threadStarts`	Total number of times a thread has been started since this JVM started.
`totalMemory`	The total amount of memory, measure in bytes, currently available for current and future objects.
`unloadedClasses`	Total number of classes unloaded since the JVM started.

JVM Garbage Collection (VMGCStats)

These statistics show how much time used by different JVM garbage collection. The primary statistics are:

Statistic	Description
`collections`	Total number of collections this garbage collector has done.
`collectionTime`	Approximate elapsed time spent doing collections by this garbage collector.

JVM Garbage Collector Memory Pools (VMMemoryPoolStats)

These statistics describe memory usage in different garbage collector memory pools. The primary statistics are:

Statistic	Description
`collectionUsageExceeded`	Total number of times the garbage collector detected that memory usage in this pool exceeded the collectionUsageThreshold.
`collectionUsageThreshold`	The collection usage threshold, measured in bytes, for this pool.
`collectionUsedMemory`	The estimated amount of used memory, measured in bytes, after that last garbage collection of this pool.
`currentCommittedMemory`	The amount of committed memory, measured in bytes, for this pool.
`currentInitMemory`	Initial memory the JVM requested from the operating system for this pool.
`currentMaxMemory`	The maximum amount of memory, measured in bytes, this pool can have.
`currentUsedMemory`	The estimated amount of used memory, measured in bytes, currently in use for this pool.

Statistic	Description
`usageExceeded`	Total number of times that memory usage in this pool exceeded the usageThreshold.
`usageThreshold`	The usage threshold, measured in bytes, for this pool.

JVM Heap Memory Usage (VMMemoryUsageStats)

Show details on how the Java heap memory is being used. The primary statistics are:

Statistic	Description
`committedMemory`	The amount of committed memory, measured in bytes, for this area.
`initMemory`	Initial memory the JVM requested from the operating system for this area.
`maxMemory`	The maximum amount of memory, measured in bytes, this area can have.
`usedMemory`	The amount of used memory, measured in bytes, for this area.

JVM Thread stats (VMThreadStats)

Show details about the JVM thread. The primary statistics are:

| Statistic | Description | |——————| ———| | | `blocked` | Total number of times this thread blocked to enter or reenter a monitor. | | `blockedTime` | Total amount of elapsed time, approximately, that this thread has spent blocked to enter or reenter a monitor. May need to be enabled by setting `-Dgemfire.enableContentionTime=true`| | `cpuTime` | Total cpu time for this thread. May need to be enabled by setting `-Dgemfire.enableCpuTime=true`. | | `inNative` | 1 if the thread is in native code. | | `lockOwner` | The thread id that owns the lock that this thread is blocking on. | | `suspended` | 1 if the thread is suspended. | | `userTime` | Total user time for this thread. May need to be enabled by setting `-Dgemfire.enableCpuTime=true`. | | `waited` | Total number of times this thread waited for notification. | | `waitedTime` | Total amount of elapsed time, approximately, that this thread has spent waiting for notification. May need to be enabled by setting `-Dgemfire.enableContentionTime=true` |

Locator (LocatorStats)

These statistics are on the VMware GemFire locator. The primary statistics are:

Statistic	Description
`ENDPOINTS_KNOWN`	Number of servers this locator knows about.
`KNOWN_LOCATORS`	Number of locators known to this locator.
`REQUEST_TIME`	Time, measured in nanoseconds, spent processing server location requests.
`REQUESTS_IN_PROGRESS`	The number of location requests currently being processed by the thread pool.
`REQUESTS_TO_LOCATOR`	Number of requests this locator has received from clients.
`RESPONSE_TIME`	Time, measured in nanoseconds, spent sending location responses to clients.

Statistic	Description
`RESPONSES_FROM_LOCATOR`	Number of responses this locator has sent to clients.
`SERVER_LOAD_UPDATES`	Total number of times a server load update has been received.

Lucene Indexes (LuceneIndexStats)

These statistics quantify the use of Lucene indexes. The primary statistics are:

Statistic	Description
`queryExecutions`	The number of Lucene queries executed on this member.
`queryExecutionTime`	The amount of time in nanoseconds spent executing Lucene queries.
`queryExecutionsInProgress`	The number of query executions currently in progress.
`queryExecutionTotalHits`	The total number of documents returned by query executions.
`repositoryQueryExecutions`	The number of Lucene repository queries executed on this member.
`repositoryQueryExecutionTime`	The amount of time in nanoseconds spent executing Lucene repository queries.
`repositoryQueryExecutionsInProgress`	The number of repository query executions currently in progress.
`repositoryQueryExecutionTotalHits`	The total number of documents returned by repository query executions.
`updates`	The number of Lucene index documents added or removed on this member.
`updateTime`	The amount of time in nanoseconds spent adding or removing documents from the index.
`updatesInProgress`	The number of index updates in progress.
`commits`	The number of Lucene index commits on this member.
`commitTime`	The amount of time in nanoseconds spent in Lucene index commits.
`commitsInProgress`	The number of Lucene index commits in progress.
`documents`	The number of documents in the index.

Off-Heap (OffHeapMemoryStats)

These statistics quantify the use of off-heap memory. The primary statistics are:

Statistic	Description
`defragmentations`	The total number of times the off-heap memory manager has invoked the defragmentation algorithm on the off-heap memory space.
`defragmentationsInProgress`	The number of defragmentation operations currently in progress.
`defragmentationTime`	The total number of nanoseconds spent running the defragmentation algorithm on off-heap memory space fragments.

Statistic	Description
`fragmentation`	This statistic gives an indication of the level of external fragmentation in the off-heap memory space by providing a ratio of the current number of fragments of free space to the largest number of fragments that could be formed from that free space. It is expressed as a percentage; the higher this value, the more fragmented the free space currently is. This statistic is 0 if the memory manager has never run its defragmentation algorithm on the off-heap space, and it is recalculated after each defragmentation.
`fragments`	The current number of fragments of free off-heap memory. This statistic is 0 if no defragmentation has ever been done, and it is updated after each defragmentation.
`freeMemory`	The number of bytes of off-heap memory that are not currently allocated.
`largestFragment`	The number of bytes in the largest fragment of memory found by the last run of the defragmentation algorithm. This statistic is updated after each defragmentation.
`maxMemory`	The number of bytes of off-heap memory initially declared.
`objects`	The number of objects currently stored in off-heap memory.
`reads`	The total number of reads of objects that are stored in off-heap memory.
`usedMemory`	The number of bytes of off-heap memory currently used for region values.

Operating System Statistics - Linux

Operating system statistics are available only for Linux operating systems.

Linux Process Performance (LinuxProcessStats)

Operating system statistics on the member's process. The primary statistics are:

Statistic	Description
`imageSize`	Size, in megabytes, of the process's image.
`rssSize`	Size, in megabytes, of the process's resident size.

Linux Operating System (LinuxSystemStats)

Operating system statistics on the member's machine. These can be used to determine total cpu, memory, and disk usage on the machine. The primary statistics are:

Statistic	Description
`allocatedSwap`	Number of megabytes of swap space that have actually been written to. Swap space must be reserved before it can be allocated.
`bufferMemory`	Number of megabytes of memory allocated to buffers.
`contextSwitches`	Total number of context switches from one thread to another on the computer. Thread switches can occur either inside of a single process or across processes. A thread switch may be caused either by one thread asking another for information, or by a thread being preempted by another, higher priority thread becoming ready to run.
`cpuActive`	Percentage of the total available time that has been used in a non-idle state.

Statistic	Description
`cpuIdle`	Percentage of the total available time that has been spent sleeping.
`cpuNice`	Percentage of the total available time that has been used to execute user code in processes with low priority.
`cpus`	Number of online CPUs (items) on the local machine.
`cpuSystem`	Percentage of the total available time that has been used to execute system (that is, kernel) code.
`cpuUser`	Percentage of the total available time that has been used to execute user code.
`freeMemory`	Number of megabytes of unused memory on the machine.
`loadAverage1`	Average number of threads in the run queue or waiting for disk I/O over the last minute.
`loadAverage15`	Average number of threads in the run queue or waiting for disk I/O over the last fifteen minutes.
`loadAverage5`	Average number of threads in the run queue or waiting for disk I/O over the last five minutes.
`loopbackBytes`	Number of network bytes sent (or received) on the loopback interface.
`loopbackPackets`	Number of network packets sent (or received) on the loopback interface.
`pagesPagedIn`	Total number of pages that have been brought into memory from disk by the operating system's memory manager.
`pagesPagedOut`	Total number of pages that have been flushed from memory to disk by the operating system's memory manager.
`pagesSwappedIn`	Total number of swap pages that have been read in from disk by the operating system's memory manager.
`pagesSwappedOut`	Total number of swap pages that have been written out to disk by the operating system's memory manager.
`physicalMemory`	Actual amount of total physical memory on the machine.
`processCreates`	The total number of times a process (operation) has been created.
`processes`	Number of processes in the computer at the time of data collection. Notice that this is an instantaneous count, not an average over the time interval. Each process represents the running of a program.
`recvBytes`	Total number of network bytes received (excluding loopback).
`recvDrops`	Total number network receives (packets) dropped.
`recvErrors`	Total number of network receive errors.
`recvPackets`	Total number of network packets received (excluding loopback).

Statistic	Description
`sharedMemory`	Number of megabytes of shared memory on the machine.
`unallocatedSwap`	Number of megabytes of swap space that have not been allocated.
`xmitBytes`	Total number of network bytes transmitted (excluding loopback).
`xmitCollisions`	Total number of network transmit collisions.
`xmitDrops`	Total number of network transmits (packets) dropped.
`xmitErrors`	Total number of network transmit errors.
`xmitPackets`	Total number of network packets transmitted (excluding loopback).

Partitioned Regions (PartitionedRegion<partitioned_region_name>Statistics)

Partitioned Region Statistics on Region Operations

These statistics track the standard region operations executed in the member. Operations can originate locally or in a request from a remote member.

Note: Unsuccessful operations are not counted in these statistics.

The primary statistics are:

Statistic	Description
`containsKeyCompleted`	Number of successful containsKey operations in this member.
`containsKeyOpsRetried`	Number of containsKey or containsValueForKey operations retried due to failures. This stat counts each retried operation only once, even if it requires multiple retries.
`containsKeyRetries`	Total number of times containsKey or containsValueForKey operations were retried. If multiple retries are required on a single operation, this stat counts them all.
`containsKeyTime`	Total time, in nanoseconds, the member spent doing containsKey operations in this member.
`containsValueForKeyCompleted`	Number of successful containsValueForKey operations in this member.
`containsValueForKeyTime`	Total time, in nanoseconds, the member spent doing containsValueForKey operations in this member.
`createOpsRetried`	Number of create operations retried due to failures. This stat counts each retried operation only once, even if it requires multiple retries.
`createRetries`	Total number of times create operations were retried. If multiple retries are required on a single operation, this stat counts them all.
`createsCompleted`	Number of successful create operations in this member.

Statistic	Description
`createTime`	Total time, in nanoseconds, the member spent doing create operations in this member.
`destroyOpsRetried`	Number of destroy operations retried due to failures. This stat counts each retried operation only once, even if it requires multiple retries.
`destroyRetries`	Total number of times destroy operations were retried. If multiple retries are required on a single operation, this stat counts them all.
`destroysCompleted`	Number of successful destroy operations in this member.
`destroyTime`	Total time, in nanoseconds, the member spent doing destroy operations in this member.
`getEntriesCompleted`	Number of get entry operations completed.
`getEntriesTime`	Total time, in nanoseconds, spent performing get entry operations.
`getOpsRetried`	Number of get operations retried due to failures. This stat counts each retried operation only once, even if it requires multiple retries.
`getRetries`	Total number of times get operations were retried. If multiple retries are required on a single operation, this stat counts them all.
`getsCompleted`	Number of successful get operations in this member.
`getTime`	Total time, in nanoseconds, the member spent doing get operations in this member.
`invalidateOpsRetried`	Number of invalidate operations retried due to failures. This stat counts each retried operation only once, even if it requires multiple retries.
`invalidateRetries`	Total number of times invalidate operations were retried. If multiple retries are required on a single operation, this stat counts them all.
`invalidatesCompleted`	Number of successful invalidate operations in this member.
`invalidateTime`	Total time, in nanoseconds, the member spent doing invalidate operations in this member.
`putOpsRetried`	Number of put operations retried due to failures. This stat counts each retried operation only once, even if it requires multiple retries.
`putRetries`	Total number of times put operations were retried. If multiple retries are required on a single operation, this stat counts them all.
`putsCompleted`	Number of successful put operations in this member.
`putTime`	Total time, in nanoseconds, the member spent doing put operations in this member.
`replyWaitMaxTime`	Longest amount of time, in milliseconds, taken to write a message and receive a reply before a forced disconnect occurs. This stat is always active regardless of the setting of the enable-time-statistics gemfire.properties setting.
`sentMessageMaxTime`	Longest amount of time, in milliseconds, taken to write a message to the network before a forced disconnect occurs. This stat is always active regardless of the setting of the enable-time-statistics gemfire.properties setting.

Partitioned Region Statistics on Partition Messages

Note: Unsuccessful operations and local operations—those that originated in this member—are not counted in these statistics.

The primary statistics are:

Statistic	Description
`partitionMessagesProcessed`	Number of region operations executed in this member at the request of other data stores for the region.
`partitionMessagesProcessingTime`	Total time, in nanoseconds, the member spent executing region operations in this member at the request of remote members.
`partitionMessagesReceived`	Number of remote requests this member received for any region operation in this member.
`partitionMessagesSent`	Number of requests this member sent for any region operation on a remote member.
`prMetaDataSentCount`	Number of times meta data refresh sent on client's request. Used with pr-single-hop functionality.

Partitioned Region Statistics on Data Entry Caching

These statistics track the pattern of data entry distribution among the buckets in this member. The primary statistics are:

Statistic	Description
`avgBucketSize`	Average number of entries for each of the primary buckets in this member.
`bucketCount`	Total number of buckets in this member.
`bucketCreationsCompleted`	Number of logical bucket creation operations requests completed after which the bucket was created.
`bucketCreationsDiscoveryCompleted`	Number of bucket creation operations requests completed after which it was discovered that the bucket was created by another member.
`bucketCreationsDiscoveryTime`	Total time, in nanoseconds, spent waiting for bucket creation requests to complete after which it was discovered that the bucket was created by another member.
`bucketCreationsTime`	Total time, in nanoseconds, spent waiting for bucket creation requests to complete after which the bucket was created.
`dataStoreBytesInUse`	The number of bytes stored in this cache for the named partitioned region.
`dataStoreEntryCount`	Total number of entries in all the buckets in this member.
`maxBucketSize`	Largest number of entries in the primary buckets in this member.
`minBucketSize`	Smallest number of entries in the primary buckets in this member.
`totalBucketSize`	Total number of entries in the primary buckets.

Partitioned Region Statistics on Redundancy

These statistics track status on partitioned region data copies. The primary statistics are:

Statistic	Description
`actualRedundantCopies`	The least current redundant number of copies for any data in this partitioned region (there may be some data that is fully redundant, but some data will have only this number of copies). This value may drop when a data store is lost or rise when a data store is added. This value may drop temporarily during partitioned region creation or destruction and then rise again. **Note:** If this value remains low, then partitioned region data is at risk and may be lost if another data store is lost. A healthy partitioned region will maintain a value equal to configuredRedundantCopies. The user should add one or more data stores if the value remains low. High-availability may result in a brief fluctuation, but it should return to a value equal to configuredRedundantCopies if there are sufficient data stores present (that is, terminating one data store will cause its data to fail over to another data store).
`configuredRedundantCopies`	This is equivalent to the PartitionAttributes.getRedundantCopies configuration that was used to create this partitioned region. This value remains unchanged for a given partitioned region.
`lowRedundancyBucketCount`	The number of buckets in this partitioned region that currently have fewer copies than the configuredRedundantCopies. This value may rise above zero when a data store is lost and return to zero when one or more data stores are added. This value may rise temporarily during partitioned region creation or destruction and then return to zero. **Note:** If this value remains above zero, then partitioned region data is at risk and may be lost if another data store is lost. This value will be above zero whenever actualRedundantCopies is less than configuredRedundantCopies. A healthy partitioned region will maintain a value of zero. The user should add one or more datstores if this value remains above zero. High-availability may result in a brief fluctuation, but it should return to zero if there are sufficient data stores present (that is, terminating one data store will cause its data to fail over to another data store).

Region Entry Eviction – Count-Based (LRUStatistics)

The entry-count least recently used (LRU) eviction mechanism records these LRUStatistics. The primary statistics are:

Statistic	Description
`entriesAllowed`	Number of entries allowed in this region.
`entryCount`	Number of entries in this region.
`lruDestroys`	Number of entry destroys triggered by an LRU.
`lruDestroysLimit`	Maximum number of entry destroys triggered by an LRU before a scan occurs.
`lruEvaluations`	Number of entries evaluated during LRU operations
`lruEvictions`	Number of total entry evictions triggered by an LRU.
`lruGreedyReturns`	Number of non-LRU entries evicted during LRU operations.

Region Entry Eviction – Heap-based eviction (HeapLRUStatistics)

The least recently used (LRU) mechanism that keeps the JVM heap size under a given set point records these LRUStatistics. The primary statistics are:

Statistic	Description
`entryBites`	The amount of memory currently used by regions configured for eviction.
`lruDestroys`	Number of entry destroys triggered by an LRU.
`lruEvaluations`	Number of entries evaluated during LRU operations
`lruEvictions`	Total number of entry evictions triggered by an LRU.
`lruGreedyReturns`	Number of non-LRU entries evicted during LRU operations.

Region Entry Eviction – Size-based (MemLRUStatistics)

The least recently used (LRU) mechanism that keeps the size of a region under a given set point records these MemLRUStatistics. The primary statistics are:

Statistic	Description
`byteCount`	Number of bytes in region
`bytesAllowed`	Total number of bytes allowed in this region.
`lruDestroys`	Number of entry destroys triggered by LRU.
`lruEvaluations`	Number of entries evaluated during LRU operations.
`lruEvictions`	Total number of entry evictions triggered by LRU.
`lruGreedyReturns`	Number of non-LRU entries evicted during LRU operations.

Server Notifications for All Clients (CacheClientNotifierStatistics)

Statistics regarding cache server operations sent to all clients.

Statistic	Description
`clientHealthMonitorRegister`	Number of clients that register.
`clientHealthMonitorUnRegister`	Number of clients that unregister.
`clientRegistrations`	Number of clients (operations) that have registered for updates.
`clientRegistrationTime`	Total time, in nanoseconds, spent doing client registrations.
`durableReconnectionCount`	Number of times the same durable client connects to the server.
`eventProcessingTime`	Total time, in nanoseconds, spent by the cache client notifier processing events.
`events`	Number of events (operations) processed by the cache client notifier.
`eventsEnqueuedWhileClientAwayCount`	Number of events enqueued for a durable client.
`queueDroppedCount`	Number of times the client subscription queue for a particular durable client is dropped.

Server Notifications for Single Client (CacheClientProxyStatistics)

Statistics regarding cache server operations and cache server client notifications sent to a single client.

Statistic	Description
`messageProcessingTime`	Total time, in nanoseconds, spent sending messages to clients.
`messageQueueSize`	Size of the operations subscription queue.
`messagesFailedQueued`	Number of client operations messages attempted but failed to be added to the subscription queue.
`messagesNotQueuedConflated`	Number of client operations messages received but not added to the subscription queue because the queue already contains a message with the message's key.
`messagesNotQueuedNotInterested`	Number of client operations messages received but not added to the subscription queue because the client represented by the receiving proxy was not interested in the message's key.
`messagesNotQueuedOriginator`	Number of client operations messages received but not added to the subscription queue, because the receiving proxy represents the client originating the message.
`messagesProcessed`	Number of client operations messages removed from the subscription queue and sent.
`messagesQueued`	Number of client operations messages added to the subscription queue.
`messagesReceived`	Number of client operations messages received.

Server-to-Client Messaging Performance (ClientSubscriptionStats)

Collected in the server, these statistics track event messages queued on the server to be sent to the client. The statistics are gathered for each client subscription queue and are incremental for the lifetime of the queue. The event messages are referred to as events in these statistics. The primary statistics are:

Statistic	Description
`eventsConflated`	Number of events conflated. If this is high, the server's dispatcher may be running slowly. This could be caused by one or more slow client.s causing blocking in their subscription queues.
`eventsExpired`	Number of events that have expired while in the subscription queue. If this is high on a secondary server, it might be that the MessageSyncInterval on the primary is set too high, causing the secondary to fall behind in event cleanup.
`eventsQueued`	Number of events placed in the subscription queue.
`eventsRemoved`	Number of events removed from the subscription queue.
`eventsRemovedByQrm`	Number of events removed based on a message sent from the primary. Only incremented while the subscription queue is in a secondary server.
`eventsTaken`	Number of events taken from the subscription queue.
`numSequenceViolated`	Number of events that had sequence ID less than or equal to the last sequence ID. The system assumes these events are duplicates and does not add them to the subscription queue. A non-zero value may indicate message loss.
`numVoidRemovals`	Number of events which were supposed to be destroyed from the subscription queue through remove but were removed by some other operation like conflation or expiration.
`threadIdentifiers`	Number of ThreadIdentifier objects (units) in the subscription queue.

Statistics Collection (StatSampler)

These statistics show how much time is spent collecting statistics. The primary statistics are:

Statistic	Description
`sampleCount`	Total number of samples taken by this sampler.
`sampleTime`	Total amount of time spent taking samples.

Transaction Reference Material

This archival section on transactions exists as a reference for understanding how to work with JTA transactions.

- **JTA Global Transactions with VMware GemFire**

  Use JTA global transactions to coordinate VMware GemFire transactions and JDBC transactions.

JTA Global Transactions with VMware GemFire

The Java Transaction API, JTA, is a standard Java interface you can use to coordinate VMware GemFire transactions and JDBC transactions globally under one umbrella.

You can use JTA global transactions to coordinate VMware GemFire transactions and JDBC transactions.

JTA provides direct coordination between the VMware GemFire cache and another transactional resource, such as a database. The parties involved in a JTA transaction include:

- The Java application, responsible for starting the global transaction
- The JTA transaction manager, responsible for opening, committing, and rolling back transactions
- The transaction resource managers, including the VMware GemFire transaction manager and the JDBC resource manager, responsible for managing operations in the VMware GemFire cache and database, respectively

Using JTA, your application controls all transactions in the same standard way, whether the transactions act on the VMware GemFire cache, a JDBC resource, or both together. When a JTA global transaction is finished, the VMware GemFire transaction and the database transaction are both complete.

When using JTA global transactions with VMware GemFire, you have two options:

- Coordinate with an external JTA transaction manager in a container (such as WebLogic or JBoss)
- Set VMware GemFire as the “last resource” while using a container (such as WebLogic or JBoss) as the JTA transaction manager

An application creates a global transaction by using `javax.transaction.UserTransaction` bound to the JNDI context `java:/UserTransaction` to start and terminate transactions. During the transaction, cache operations are done through VMware GemFire as usual.

Note: See the Java documentation for more information on topics such as JTA, `javax.transaction`, committing and rolling back global transactions, and the related exceptions.

- **Coordinating with External JTA Transactions Managers**

  VMware GemFire can work with the JTA transaction managers of several containers like JBoss, WebLogic, GlassFish, and so on.

- **Using VMware GemFire as the “Last Resource” in a Container-Managed JTA Transaction**

  The “last resource” feature in certain third party containers such as WebLogic allow the use one non-XAResource (such as VMware GemFire) in a transaction with multiple XAResources while ensuring consistency.

- **Behavior of VMware GemFire Cache Writers and Loaders Under JTA**

  When VMware GemFire participates in a global transactions, you can still have VMware GemFire cache writers and cache loaders operating in the usual way.

- **Turning Off JTA Transactions**

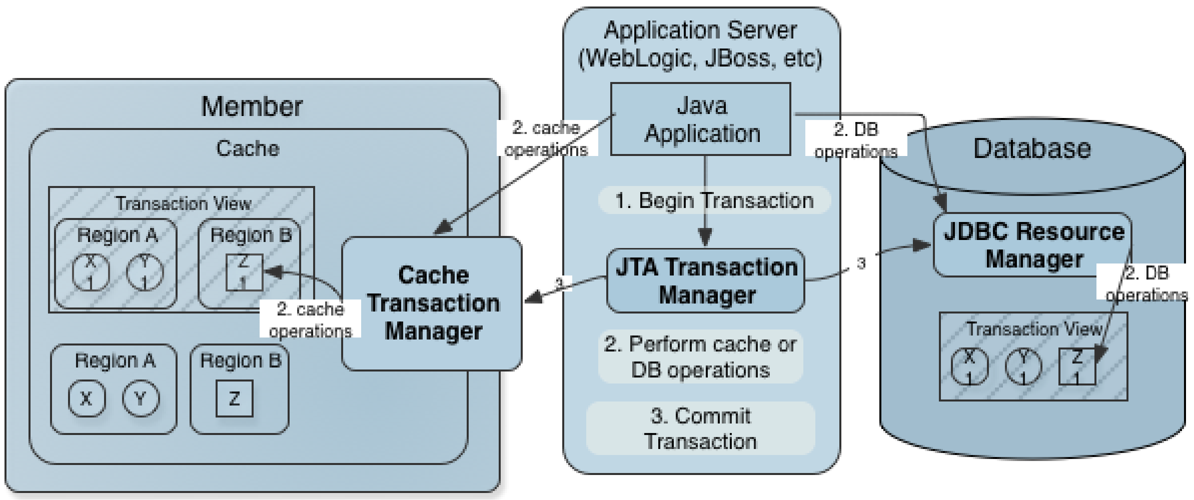
You can configure regions to not participate in any JTA global transaction.

Coordinating with External JTA Transaction Managers

VMware GemFire can work with the JTA transaction managers of several containers such as JBoss, WebLogic, GlassFish, and so on.

At startup VMware GemFire looks for a TransactionManager (`javax.transaction.TransactionManager`) that has been bound to its JNDI context. When VMware GemFire finds such an external transaction manager, all VMware GemFire region operations (such as get and put) will participate in global transactions hosted by this external JTA transaction manager.

This figure shows the high-level operation of a JTA global transaction whose resources include a VMware GemFire cache and a database.



An externally coordinated JTA global transaction is run in the following manner:

1. Each region operation looks up for presence of a global transaction. If one is detected, then a VMware GemFire transaction is started automatically, and we register a `javax.transaction.Synchronization` callback with the external JTA transaction manager.
2. At transaction commit, VMware GemFire gets a `beforeCommit()` callback from the external JTA transaction manager. VMware GemFire does all locking and conflict detection at this time. If this fails, an exception is thrown back to JTA transaction manager, which then cancels the transaction.
3. After a successful `beforeCommit()`callback, JTA transaction manager asks other data sources to commit their transaction.
4. VMware GemFire then gets a `afterCommit()` callback in which changes are applied to the cache and distributed to other members.

You can deactivate JTA in any region that should not participate in JTA transactions. See Turning Off JTA Transactions.

How to Run a JTA Transaction Coordinated by an External Transaction Manager

Use the following procedure to run a VMware GemFire global JTA transaction coordinated by an external JTA transaction manager.

1. **Configure the external data sources in the external container.** Do not configure the data sources in cache.xml . They are not guaranteed to get bound to the JNDI tree. 2.

   Configure VMware GemFire for any necessary transactional behavior in the `cache.xml` file. For example, enable `copy-on-read` and specify a transaction listener, as needed. See Copy on Read Behavior.

2. Make sure that JTA transactions are enabled for the regions that will participate in the transaction. See Turning Off JTA Transactions for details. 4.

   Start the transaction through the external container.

3. Initialize the VMware GemFire cache. VMware GemFire will automatically join the transaction. 6.

   Execute operations in the cache and the database as usual.

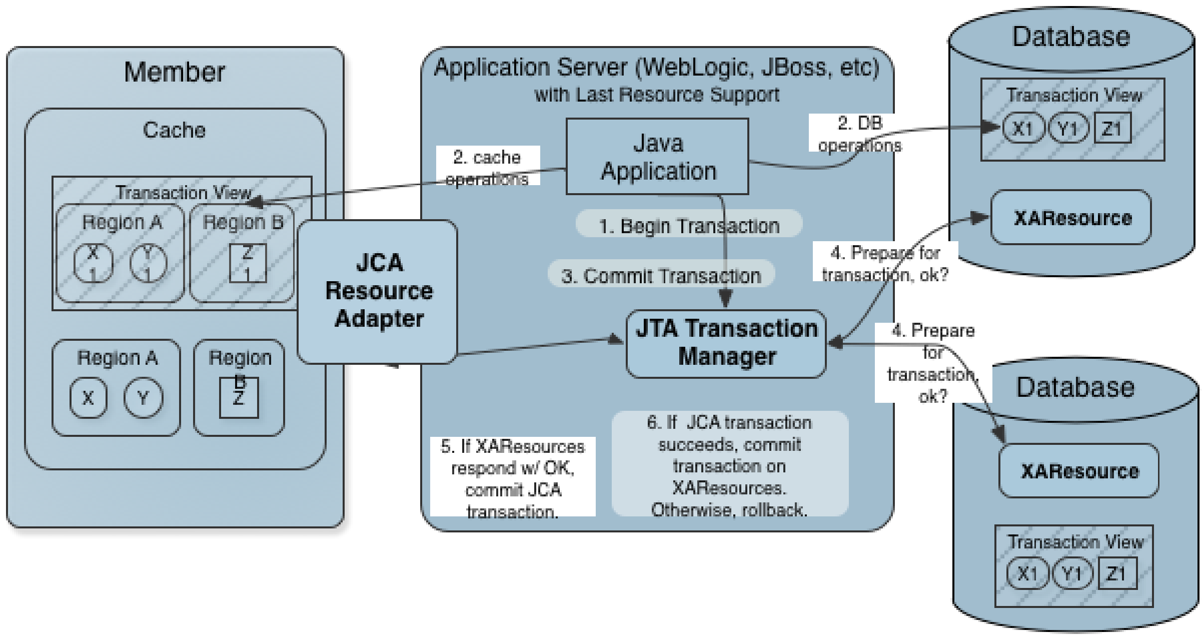
4. Commit the transaction through the external container.

Using VMware GemFire as the “Last Resource” in a Container-Managed JTA Transaction

The “last resource” feature in certain third party containers such as WebLogic allow the use of one non-XAResource (such as VMware GemFire) in a transaction with multiple XAResources while ensuring consistency.

In the previous two JTA transaction use cases, if the VMware GemFire member fails after the other data sources commit but before VMware GemFire receives the `afterCommit` callback, VMware GemFire and the other data sources may become inconsistent. To prevent this from occurring, you can use the container’s “last resource optimization” feature, with VMware GemFire set as the “last resource”. Using VMware GemFire as the last resource ensures that in the event of failure, VMware GemFire remains consistent with the other XAResources involved in the transaction.

To accomplish this, the application server container must use a JCA Resource Adapter to accomodate VMware GemFire as the transaction’s last resource. The transaction manager of the container first issues a “prepare” message to the participating XAResources. If the XAResources all accept the transaction, then the manager issues a “commit” instruction to the non-XAResource (in this case, VMware GemFire). The non-XAResource (in this case, VMware GemFire) participates as a local transaction resource. If the non-XAResource fails, then the transaction manager can rollback the XAResources.


Member
Cache
Transaction View
Region A
Region B
Region A
Region B
JCA Resource Adapter
Application Server (WebLogic, JBoss, etc)
with Last Resource Support
Java Application
2. cache operations
1. Begin Transaction
3. Commit Transaction
JTA Transaction Manager
2. DB operations
4. Prepare for transaction, ok?
4. Prepare for transaction ok?
5. If XAResources respond w/ OK, commit JCA transaction.
6. If JCA transaction succeeds, commit transaction on XAResources. Otherwise, rollback.
Database
Transaction View
XAResource
Database
XAResource
Transaction View

How to Run JTA Transactions with VMware GemFire as a "Last Resource"

1. Locate the version-specific `geode-jca` RAR file within the `lib` directory of your VMware GemFire installation.
2. Add your container-specific XML file to the `geode-jca` RAR file.

1. Create a container-specific resource adapter XML file named <container>-ra.xml. For example, an XML file for a WebLogic resource adapter XML file might look something like this: ``` pre GFE JCA gfe/jca ```
2. Create a folder named `META-INF`, and place the container-specific XML file inside the directory. For example, the folder structure would look like this: ``` pre META-INF/weblogic-ra.xml ```
3. Navigate to the directory above the `META-INF` folder and execute the following command, with appropriate substitutions for path and file names: ``` pre $ jar -uf /path/to/lib/geode-jca-X-X-X.rar META-INF/weblogic-ra.xml ```

1. Make sure that the `geode-dependencies.jar` is accessible in the CLASSPATH of the JTA transaction coordinator container.
2. Deploy the version-specific `geode-jca` RAR file on the JTA transaction coordinator container. When deploying the file, you specify the JNDI name and so on.
3. Configure VMware GemFire for any necessary transactional behavior. Enable `copy-on-read` and specify a transaction listener, if you need one. See Copy on Read Behavior.
4. Get an initial context through `org.apache.geode.cache.GemFireCache.getJNDIContext`. For example:

```
Context ctx = cache.getJNDIContext();
```

This returns `javax.naming.Context` and gives you the JNDI associated with the cache. The context contains the `TransactionManager`, `UserTransaction`, and any configured JDBC resource manager.

5. Start and commit the global transaction using the `UserTransaction` object rather than with VMware GemFire's `CacheTransactionManager`.

```
UserTransaction txManager = (UserTransaction)ctx.lookup("java:/UserTransactio
n");
```

6. Obtain a VMware GemFire connection.

```
GFConnectionFactory cf = (GFConnectionFactory) ctx.lookup("gfe/jca");

//This step of obtaining connection is what begins the
//LocalTransaction.
//If this is absent, GFE operations will not be part of any
//transaction
GFConnection gemfireConn = (GFConnection)cf.getConnection();
```

Behavior of VMware GemFire Cache Writers and Loaders Under JTA

When VMware GemFire participates in a global transactions, you can still have VMware GemFire cache writers and cache loaders operating in the usual way.

For example, in addition to the transactional connection to the database, the region could also have a cache writer and cache loader configured to exchange data with that same database. As long as the data source is transactional, which means that it can detect the transaction manager, the cache writer and cache loader participate in the transaction. If the JTA rolls back its transaction, the changes made by the cache loader and the cache writer are rolled back. For more on transactional data sources, see the discussion of XAPooledDataSource and ManagedDataSource in Configuring Database Connections Using JNDI.

If you are using a VMware GemFire cache or transaction listener with global transactions, be aware that the EntryEvent returned by a transaction has the VMware GemFire transaction ID, not the JTA transaction ID.

Turning Off JTA Transactions

You can configure regions to not participate in any JTA global transaction.

The `ignore-jta` region attribute is a boolean that tells the cache to ignore any in-progress JTA transactions when performing cache operations. It is primarily used for cache loaders, cache writers, and cache listeners that need to perform non-transactional operations on a region, such as caching a result set. It is set per region, so some regions can participate in JTA transactions, while others avoid participating in them. This example sets the `ignore-jta` region attribute in the `cache.xml` file.

cache.xml:

```
<region name="bridge_region">
   <region-attributes scope="local" ignore-jta="true" statistics-enabled="true"/>
       <cache-writer> . . . </cache-writer>
    </region-attributes>
</region>
```

API:

Using the API, you can turn off JTA transactions using `RegionFactory` and its method `setIgnoreJTA(boolean)`. The current setting for a region can be fetched from a region's `RegionAttributes` by using the `getIgnoreJTA` method.

Experimental Features

VMware GemFire includes these features to allow interested developers to experiment with their use in their own development systems. Feedback will help drive development of these features, and they may be supported in future versions of the product.

These features may change in or be removed from future versions of the product.

- **Cluster Management Service**

  The cluster management service enables you to change the configuration of the cluster (for example, create/destroy regions, indexes, or gateway receivers/senders), and have these changes replicated on all the applicable servers and persisted in the cluster configuration service.

- **Redis Adapter**

  The Redis adapter allows Geode to function as a drop-in replacement for a Redis data store, letting Redis applications take advantage of Geode's scaling capabilities without changing their client code. Redis clients connect to a Geode server in the same way they connect to a Redis server, using an IP address and a port number.

- **Automated Rebalancing of Partitioned Region Data**

  The automated rebalance feature triggers a rebalance operation based on a time schedule.

Cluster Management Service

Note: This feature is experimental and is subject to change in future releases of VMware GemFire.

The cluster management service enables you to change the configuration of the cluster (for example, create/destroy regions, indexes, or gateway receivers/senders), and have these changes replicated on all the applicable servers and persisted in the cluster configuration service.

With the cluster management service, you can:

- Create regions/indexes on the fly.
- Persist the configuration and apply it to the cluster, so when a new node joins, it has the configuration, and when the server restarts, it has the configuration.
- Obtain a consistent view of the current configuration
- Apply a change to all elements of the cluster in the same way
- Be able to change the configuration in one place
- Obtain this configuration without being on the cluster
- Invoke the configuration service from the server or from a client

The cluster management service is documented in the Geode Wiki: Cluster Management Service. The Wiki write-up offers separate choices for the cluster management service implementation on each Geode version that supports it.

The cluster management service is available as a REST API and as a Java API.

Cluster Management Service REST API

The cluster management service REST API adheres to standard REST semantics, so users can use POST, PATCH, DELETE, and GET to create, update, delete, and read, respectively.

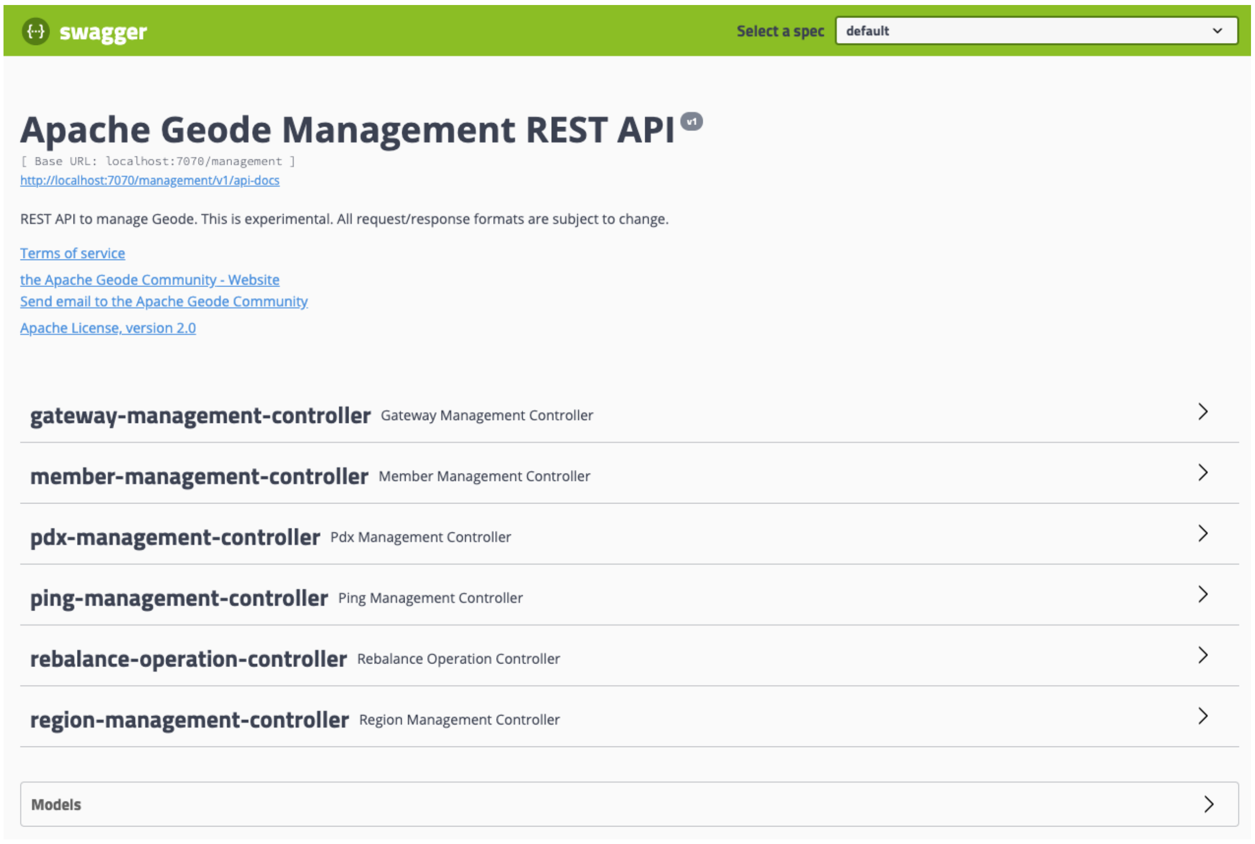
This version of VMware GemFire uses the 1.12.0 Management REST API. As mentioned, the feature is experimental and still undergoing revision.

For a definitive description of the cluster management service for your version of VMware GemFire, you can use the Swagger™ framework. To use the Swagger UI to browse the cluster management service API:

1. Using `gfsh`, start a locator using the default settings.
2. Open a browser and enter the following URL:

```
http://localhost:7070/management/docs
```

This produces a browsable description of the cluster management service API:



Apache Geode Management REST API v1

[Base URL: localhost:7070/management]
http://localhost:7070/management/v1/api-docs

REST API to manage Geode. This is experimental. All request/response formats are subject to change.

Terms of service
the Apache Geode Community - Website
Send email to the Apache Geode Community
Apache License, version 2.0

gateway-management-controller Gateway Management Controller
member-management-controller Member Management Controller
pdx-management-controller Pdx Management Controller
ping-management-controller Ping Management Controller
rebalance-operation-controller Rebalance Operation Controller
region-management-controller Region Management Controller

Models

Cluster Management Service Java API

The cluster management service is available to Java clients via a Java API. To enable the cluster management service Java API, include `geode-management.jar` in your client classpath. See the Cluster Management Service Java Client topic on the Geode Wiki for more details.

VMware GemFire Micrometer

VMware GemFire uses Micrometer to provide an interface for emitting user-focused metrics, instrumented in code, to a variety of consuming Application Performance Monitoring (APM) tools. VMware GemFire has Micrometer instrumentation added to a subset of information available, which can be easily emitted from a Meter Registry.

By default, Micrometer is enabled and can emit metrics to any meter registry. However, time-based meters are available only when the `time-statistics` property is enabled.

The API is currently marked experimental. The `MetricsSession` and `MetricsPublishingService` interfaces may change.

- **Micrometer configuration and publishing**

  Using Micrometer may require configuration. To emit Micrometer metrics to one of the supported APM tools, or to a custom format or tool, you must use a Meter Registry and accompanying publishing service.

- **Micrometer meters and tags**

  VMware GemFire contains a set of meters – instrumented statistics – that provide information about each cluster member. Each member within a VMware GemFire cluster can emit a set of meters of different types, timers, gauges, and counters specific to the member. Each meter has a set of common tags, a name/value pair, that help users identify where a meter is located in the cluster. Meter-specific tags also exist.

Configuration and Publishing

Meter configuration

Out of the box VMware GemFire is instrumented and can emit meters with a properly created Meter Registry.

However, any meters that are timers based on the system clock time, similar to the time statistics, they will obey the existing `enable-time-statistics` geode.properties setting, see **Setting up Statistics**.

Publishing metrics using a meter registry

In order to emit metrics to an Application Performance Monitor (APM) or other such tool that can store or display metrics, a meter registry is required. Micrometer implements many different meter registries as project imports, a list can be found here: https://micrometer.io/docs.

As a simple example, below describes how you might create a publishing service utilizing the `MetricsSession` and `MetricsPublishingService` interface in VMware GemFire.

Here is a example class that would enable Prometheus metrics to be emitted:

```
public class SimpleMetricsPublishingService implements MetricsPublishingService {

  private static final String PORT_PROPERTY = "prometheus.metrics.port";
  private static final int DEFAULT_PORT = 0; // If no port specified, use any port
  private static final String HOSTNAME = "localhost";
  private static final int PORT = getInteger(PORT_PROPERTY, DEFAULT_PORT);

  private static Logger LOG = getLogger(SimpleMetricsPublishingService.class);

  private final int port;
  private PrometheusMeterRegistry registry;
  private HttpServer server;

  public SimpleMetricsPublishingService() {
    this(PORT);
  }

  public SimpleMetricsPublishingService(int port) {
    this.port = port;
  }

  @Override
  public void start(MetricsSession session) {
    registry = new PrometheusMeterRegistry(DEFAULT);
    session.addSubregistry(registry);

    InetSocketAddress address = new InetSocketAddress(HOSTNAME, port);
    server = null;
    try {
      server = HttpServer.create(address, 0);
    } catch (IOException thrown) {
      LOG.error("Exception while starting " + getClass().getSimpleName(), thrown);
    }
    HttpContext context = server.createContext("/");
    context.setHandler(this::requestHandler);
    server.start();

    int boundPort = server.getAddress().getPort();
    LOG.info("Started {} http://{}:{}/", getClass().getSimpleName(), HOSTNAME, boundPo
rt);
  }

  private void requestHandler(HttpExchange httpExchange) throws IOException {
    final byte[] scrapeBytes = registry.scrape().getBytes();
    httpExchange.sendResponseHeaders(200, scrapeBytes.length);
    final OutputStream responseBody = httpExchange.getResponseBody();
    responseBody.write(scrapeBytes);
    responseBody.close();
  }

  @Override
  public void stop(MetricsSession session) {
    session.removeSubregistry(registry);
    registry = null;
    server.stop(0);
  }
}
```

To make your service discoverable, add the following provider-configuration file in the resource directory of your publishing service jar file:

```
META-INF/services/org.apache.geode.metrics.MetricsPublishingService
```

Add a line inside the file indicating the fully qualified class name of your implementation:

```
my.domain.SimpleMetricsPublishingService
```

Add Your jar File to the classpath When You Start a Server or Locator

To add your metrics publishing service to a server or locator, add your jar file to the classpath when you start the server or locator via `GFSH` and specify the `prometheus.metrics.port` listed in SimpleMetricsPublishingService:

```
gfsh>create locator --name my-locator --classpath=<path-to-my-jar-file>/my.jar --J=-Dp
rometheus.metrics.port=9914

gfsh>create server --name my-server --classpath=<path-to-my-jar-file>/my.jar --J=-Dpro
metheus.metrics.port=9915
```

Alternatively, you can add your jar file to the extensions directory in your Geode installation and only specify the `prometheus.metrics.port`. Then `GFSH` will add your jar file to the classpath whenever it creates a server or locator.

Micrometer Meters and Tags

Micrometer Meters

Micrometer meters are similar to VMware GemFire statistics in that they have different types. VMware GemFire uses the three main meter types: counter, gauge, and timer.

- A counter is a numerical value that only goes up.
- A gauge is a numerical value that can go up or down.
- A timer is a counter combined with an ever growing summary of the time each thing being counted has taken to finish.

There are other types of meters available in Micrometer, but they are not currently being used in VMware GemFire.

Every meter in Micrometer has the following attributes, but may also have more than these depending on the meter type:

- `Name`: a lowercase, 'dot' delimited name
- `Description`: a short text description of the meter
- `Value`: the value of the meter, always a number
- `Units`: the unit of the value
- `Tags`: key/value string pairs to define facts about the meter

Meters supplied by Micrometer

VMware GemFire has been instrumented with a set of Micrometer provided meters, called binders, which contain one or many individual meters with pre-defined types and tags. The following are a subset of the Micrometer binders that have been added to VMware GemFire:

JVM specific

- JvmGcMetrics
- JvmMemoryMetrics
- JvmThreadMetrics

Operating System specific

- ProcessorMetrics
- UptimeMetrics
- FileDescriptorMetrics

Meters specific to VMware GemFire

There are VMware GemFire specific meters that have been instrumented to allow developers to monitor certain aspects of the VMware GemFire cluster. The namespace used for these metrics includes the Apache Geode (e.g. geode.*) product name as a means to distinguish this meter from others similar to it in downstream systems. The following meters have been added to VMware GemFire:

Meter:	**geode.cache.entries**
Type:	Gauge
Description:	The current count of entries in the cache for this member (locators will not have this metric)
Tag:	region: The name of the region associated with the count of entries
Tag:	data.policy: The data policy described for these regions, see RegionShortcuts for Peers and Servers

Meter:	**geode.function.executions**
Type:	Timer
Description:	A total time and count for any execution that completed without errors of this function on this member.
Tag:	Function: The name or class of the function
Tag:	Succeeded: true/false

Meter:	**geode.gateway.receiver.events**
Type:	Counter
Description:	The total number of events received by this GatewayReceiver

Meter:	**geode.cache.gets**
Type:	Timer

Meter:	geode.cache.gets
Description:	A total time and count for any get operation performed against the cache server for this member. Get operations that circumvent the cache server, such as REST calls, gfsh operations, and peer-to-peer operations, are not counted. Locators will not have this metric.
Tag:	region: The name of the region associated with the count of entries
Tag:	result: hit/miss

Common tags

VMware GemFire meters all contain tags, also referred to as labels or dimensions, depending upon which Application Performance Monitoring (APM) tool is being utilized. Micrometer offers a concept of `common` tags that exist on every meter within a meter registry. In VMware GemFire, the following common tags have been provided to add context to each meter:

- `cluster`: The distributed system id associated with this member
- `host`: The hostname the member exists on, could be the same as another if collocated
- `member`: the member name provided at startup time
- `member.type`: locator, server, locator-server

Please submit a pull request to Apache Geode if there are more metrics you would like to see added to VMware GemFire.

Redis Adapter

Note: This feature is experimental and is subject to change in future releases of VMware GemFire.

The VMware GemFire Redis adapter allows VMware GemFire to function as a drop-in replacement for a Redis data store, letting Redis applications take advantage of VMware GemFire's scaling capabilities without changing their client code. Redis clients connect to a VMware GemFire server in the same way they connect to a Redis server, using an IP address and a port number.

- **Using the Redis Adapter**
- **How the Redis Adapter Works**
- **Advantages of VMware GemFire over a Redis Server**

Using the Redis Adapter

To use the Redis Adapter, you will need three pieces of information:

1. The port number through which clients will communicate
2. The IP address of the host where the server is to reside
3. A choice of which attributes you will use for a VMware GemFire partitioned region

The IP address and port number should be the same ones coded in the Redis clients.

In order to take advantage of VMware GemFire's scaling capabilities, you should specify the VMware GemFire region as one of the types that use the PARTITION data policy. PARTITION is the default. Other possibilities include PARTITION_REDUNDANT and PARTITION_PERSISTENT. (See "Region Shortcuts Quick Reference" for a complete list.)

To implement a VMware GemFire instance using the Redis Adapter:

1. Install VMware GemFire on the system where the server is to reside.
2. Use gfsh to start a VMware GemFire server, specifying the three configuration options described above:
   - Use `--redis-port` to specify the port. This parameter is required – the VMware GemFire server will listen on this port for Redis commands.
   - Use `--redis-bind-address` to specify the IP address of the server host. This parameter is optional. If not specified, the default is determined from the /etc/hosts file.
   - Use `--J=-Dgemfireredis.regiontype` to specify the region type. This parameter is optional. If not specified, regiontype is set to PARTITION.

For example:

```
gfsh> start server --name=server1 --redis-bind-address=localhost \
 --redis-port=11211 --J=-Dgemfireredis.regiontype=PARTITION_PERSISTENT
```

Redis clients can then connect to the server at localhost:11211.

How the Redis Adapter Works

The VMware GemFire Redis Adapter supports all Redis data structures, including

- String
- List
- Hash
- Set
- SortedSet
- HyperLogLog

In VMware GemFire these data structures are implemented using partitioned regions. In most cases, VMware GemFire allocates one partitioned region for each data structure. For example, each Sorted Set is allocated its own partitioned region, in which the key is the user data and the value is the user-provided score, and entries are indexed by score. The two exceptions to this design are data types String and HyperLogLog. All Strings are allocated to a single partitioned region. Similarly, all HyperLogLogs are allocated to a single region. Regions use VMware GemFire's OQL and indexes.

The VMware GemFire Redis Adapter supports all Redis commands for each of the Redis data structures. (See the Javadocs for the GemFireRedisServer class for a detailed list.) The VMware GemFire server's responses to Redis commands are identical to those of a Redis server with the following exceptions, resulting from VMware GemFire's more extensive partitioning model:

- Any command that removes keys and returns a count of removed entries will return a count of how many entries have been removed from the local vm, rather than a total count of items removed across all members. However, all entries will be removed.

- Any command that returns a count of newly set members has an unspecified return value. The command will work just as the Redis protocol states, but the count will not necessarily reflect the number set compared to the number overridden.
- Transactions work just as they would on a Redis instance; they are local transactions. Transactions cannot be executed on data that is not local to the executing server, that is on a partitioned region in a different server instance, or that is on a persistent region that does not have transactions enabled. Also, you cannot watch or unwatch keys, as all keys within a VMware GemFire transaction are watched by default.

Advantages of VMware GemFire over a Redis Server

VMware GemFire's primary advantage is its **scalability**. While the Redis server is single threaded, VMware GemFire supports high concurrency. Many Redis clients can execute commands on the VMware GemFire server simultaneously.

VMware GemFire supports **stored procedures**, which can execute on the server and report results to the requesting client.

VMware GemFire architecture and management features help detect and resolve **network partitioning** problems without explicit management on the part of the Redis client.

VMware GemFire **WAN replication** allows the data store to expand horizontally, across physically distant sites, while maintaining data consistency.

Automated Rebalancing of Partitioned Region Data

Note: This feature is experimental and is subject to change in future releases of VMware GemFire.

Automated rebalance triggers a rebalance (see Rebalancing Partitioned Region Data) operation based on a time schedule. At the scheduled intervals, the balance of the partitioned regions is evaluated based on configured criteria. One criterion is a minimum threshhold for number of bytes that would be transferred if the rebalance takes place. The other criterion uses the ratio of the number of bytes that would be transferred to the total number of bytes in the regions. If the evaluation indicates the system is out of balance, the rebalance transfer is initiated.

To enable automated rebalance, specify the rebalance manager in the `<initializer>` attribute within the `<cache>` configuration of the `cache.xml` file:

```
<class-name> org.apache.geode.cache.util.AutoBalancer </class-name>
```

The time schedule that triggers an evaluation and possible rebalance uses a cron-based specification in the `<initializer>` attribute within the `<cache>` configuration of the `cache.xml` file. This scheduling specification is required. Specify the cron expression in the Spring format. This example specification triggers each Saturday at 3am:

```
<parameter name="schedule"> 0 0 3 ? * SAT </parameter>
```

This example specification triggers once each day at 4am:

```
<parameter name="schedule"> 0 0 4 * * ?</parameter>
```

The automated rebalance specifications that specify criteria for triggering the rebalance are optional and have reasonable default values.

One criterion is a minimum number of bytes that would be transferred if the rebalance were to take place. The specification is in units of bytes; here is the specification for the default value of 100MB:

```
<parameter name="minimum-size"> 104857600</parameter>
```

Another criterion represents the ratio of bytes that would be transferred to the total number of bytes in the partitioned regions, represented as an integer percentage. The default is 10 percent. This example specificies 15 percent:

```
<parameter name="size-threshold-percent"> 15 </parameter>
```

Glossary

This glossary defines terms used in VMware GemFire documentation.

ACK wait threshold

A time-to-wait for message acknowledgment between system members.

administrative event

See event.

API

Application Programming Interface. VMware GemFire provides APIs to cached data for Java applications.

application program

A program designed to perform a specific function directly for the user or, in some cases, for another application program. VMware GemFire applications use the VMware GemFire application programming interfaces (APIs) to modify cached data.

attribute

Querying: A named member of a data object. The public fields and methods of an object may be accessed as attributes in the context of a query.

Region: See region attributes.

attribute path

A sequence of attributes separated by a dot (.), applied to objects where the value of each attribute is used to apply the next attribute.

blocking

A behavior associated with synchronization functions. Blocking behavior is exhibited as waiting for a signal to proceed, regardless of how long it takes. See also timeout.

cache

In-memory VMware GemFire data storage created by an application or cache server for data storage, distribution, and management. This is the point of access for Java applications for all caching features, and the only view of the cache that is available to the application. Cache creation creates a connection to the cluster. See also local and remote.

cache-local

Residing or occurring in the local cache.

cache.xml

Common name for the XML file that declares the initial configuration of a cache. This file is used to customize the behavior of the VMware GemFire cache server process and can be used by any Java application. Applications can also configure the cache through the VMware GemFire Java APIs. You can give this file any name.

cache event

See event.

cache listener

User-implemented plug-in for receiving and handling region entry events. A region's cache listener is called after an entry in the local cache is modified. See also cache writer.

cache loader

User-implemented plug-in for loading data into a region. A region's cache loader is used to load data that is requested of the region but is not available in the cluster. For a distributed region, the loader that is used can be in a different cache from the one where the data-request operation originated. See also netSearch and netLoad.cache misses, where a requested key is not present or has a null value in the local cache.

cache miss

The situation where a key's value is requested from a cache and the requested key is not present or has a null value. VMware GemFire responds to cache misses in various ways, depending on the region and system configuration. For example, a client region goes to its servers to satisfy cache misses. A region with local scope uses its data loader to load the value from an outside data source, if a loader is installed on the region.

cache server

A long-lived, configurable VMware GemFire cluster member process that can service client connections.

cache transaction

A native VMware GemFire transaction, managed by VMware GemFire and not by JTA. This type of transaction operates only on data available from the VMware GemFire cache in the local member. See also JTA and global transaction.

cache writer

User-implemented plug-in intended for synchronizing the cache with an outside data source. A region's cache writer is a synchronous listener to cache data events. The cache writer has the ability to cancel a data modification. See also cache listener and netWrite.

client

A VMware GemFire application that is configured as a standalone cluster member, with regions configured as client regions. Client configuration uses the <client-cache> `cache.xml` element and the ClientCache API.

client region

A VMware GemFire cache region that is configured to go to one or more VMware GemFire servers, in a separate VMware GemFire cluster, for all data distribution activities. Among other things, client regions go to servers to satisfy cache misses, distribute data modifications, and to run single queries and continuous queries.

cluster configuration service

The cluster configuration service saves cluster configurations created by gfsh commands to the locators in a cluster and distributes the configurations to members of the cluster.

collection

Used in the context of a query for a group of distinct objects of homogeneous type, referred to as elements. Valid collections include the `java.util.Collection` as well as Set, Map, List, and arrays. The elements in a collection can be iterated over. Iteration over a Map traverses its entries as instances of Map.Entry. A region can also be treated as a collection of its values.

commit

A transactional operation that merges a transaction's result into the cache. Changes are made in an “all or none” fashion. Other changes from outside the current transaction are kept separate from those being committed.

concurrency-level

Region attribute that specifies an estimate of the number of threads ever expected to concurrently modify values in the region. The actual concurrency may vary; this value is used to optimize the allocation of system resources.

conflation

Combining entries in a message queue for better performance. When an event is added to queue, if a similar event exists in the queue, there are two ways to conflate the events. One way is to remove the existing entry from wherever it resides in the queue, and add the new entry to the end of the queue. The other way is to replace the existing entry with the new entry, where it resides in the queue, and add nothing to the end of the queue. In VMware GemFire, region entry update events, server events going to clients, and gateway sender events going to remote clusters can all be conflated.

connection

The connection used by an application to access a VMware GemFire system. A Java application connects to its VMware GemFire cluster when it creates its cache. The application must connect to a cluster to gain access to the VMware GemFire functionalities. A client connects to a running VMware GemFire server to distribute data and events between itself and the server tier. These client connections are managed by server connection pools within the client applications. Gateway senders connect to a remote gateway receiver to distribute data events between sites.

consumer

VMware GemFire member process that receives data and/or events from other members. Peer consumers are often configured with replicated regions, so all changes in the cluster arrive into the local cache. Client consumers can register subscriptions with their servers so that updates are automatically forwarded from the server tier. See producer.

coordinator

The member of the cluster that sends out membership views. This is typically the locator in VMware GemFire.

data accessor

In the context of a region, a member configured to use a region, but not store any data for it in the member's local cache. Common use cases for data accessors are thin clients, and thin producer and consumer applications. Accessors can put data into the region and receive events for the region from remote members or servers, but they store no data in the application. See also data store.

data entry

See entry.

data fabric

Also referred to as an in-memory data grid or an enterprise data fabric. VMware GemFire is an implementation of a data fabric. A data fabric is a distributed, memory-based data management platform that uses cluster-wide resources – memory, CPU, network bandwidth, and optionally local disk – to manage application data and application logic (behavior). The data fabric uses dynamic replication and data partitioning techniques to offer continuous availability, very high performance, and linear scalability for data intensive applications, all without compromising on data consistency even when exposed to failure conditions.

data-policy

Region attribute used to determine what events the region receives from remote caches, whether data is stored in the local cache, and whether the data is persisted to disk. For disk persistence, writes are performed according to the cache disk-store configuration.

data region (region)

A logical grouping of data within a cache. Regions usually contain data entries (see entry). Each region has a set of region attributes governing activities such as expiration, distribution, data loading, events, and capacity control. In addition, a region can have an application-defined user attribute.

data store

In the context of a region, a member configured to store data for the region. This is used mostly for partitioned regions, where data is spread across the cluster among the data stores. See also data accessor.

deadlock

A situation in which two or more processes are waiting indefinitely for events that will never occur.

destroy

Distributed: To remove a cached object across the distributed cache.

Local: To remove a cached object from the local cache only.

disk region

A persistent region.

disk-store

Cache element specifying location and write behavior for disk storage. Used for persistence and overflow of data. The cache can have multiple disk stores, which are specified by name for region attributes, client subscription queues (for servers), and WAN gateway sender queues.

distributed cache

A collection of caches spread across multiple machines and multiple locations that functions as a single cache for the individual applications.

distributed system

One or more VMware GemFire system members or clusters that have been configured to communicate cache events with each other, forming a single, logical system.

distributed-ack scope

Data distribution setting that causes synchronous distribution operations, which wait for acknowledgment from other caches before continuing. Operations from multiple caches can arrive out of order. This scope is slower but more reliable than distributed-no-ack.

distributed-no-ack scope

Data distribution setting that causes asynchronous distribution operations, which return without waiting for a response from other caches. This scope produces the best performance, but is prone to race conditions.

entry

A data object in a region consisting of a key and a value. The value is either null (invalid) or a Java object. A region entry knows what region it is in. An entry can have an application-defined user attribute. See also region data, entry key, and entry value.

entry key

The unique identifier for an entry in a region.

entry value

The data contained in an entry.

event

An action recognized by the VMware GemFire system members, which can respond by executing callback methods. The VMware GemFire API produces two types of events: cache events for detail-level management of applications with data caches and administrative events for higher-level management of the cluster and its components. An operation can produce administrative events, cache events, or both.

eviction-attributes

Region attribute that causes the cache to limit the size of the region by removing old entries to make space for new ones.

expiration

A cached object expires when its time-to-live or idle timeout counters are exhausted. A region has one set of expiration attributes for itself and one set for all of its entries.

expiration action

The action to be taken when a cached object expires. The expiration action specifies whether the object is to be invalidated or destroyed and whether the action is to be performed only in the local cache or throughout the cluster. A destroyed object is completely removed from the cache. A region is invalidated by invalidating all entries contained in the region. An entry is invalidated by having its value marked as invalid. Region.getEntry.getValue returns null for an invalid entry.

In VMware GemFire, expiration attributes are set at the region level for the region and at the entry level for entries. See also idle timeout and time-to-live.

factory method

An interface for creating an object which at creation time can let its subclasses decide which class to instantiate. The factory method helps instantiate the appropriate subclass by creating the correct object from a group of related classes.

forced disconnect

Forcible removal of a member from membership without the member's consent.

gateway receiver

A gateway receiver defines connection information for receiving region events that were distributed from a gateway sender in a multi-site deployment.

gateway sender

A gateway sender defines a single remote cluster site and an associated queue for distributing region events in a multi-site deployment.

gemfire.properties

Common name for the file used for cluster configuration, including system member connection and communication behavior, logging and statistics files and settings, and security settings. Applications can also configure the cluster through the VMware GemFire Java APIs. You can give this file any name.

global scope

Data distribution setting that provides locking across the cluster for load, create, put, invalidate, and destroy operations on the region and its entries. This scope is the slowest, but it guarantees consistency across the cluster.

global transaction

A JTA-controlled transaction in which multiple resources, such as the VMware GemFire cache and a JDBC database connection, participate. JTA coordinates the completion of the transaction with each of the transaction's resources. See also JTA and cache transaction.

HTTP

World Wide Web's Hypertext Transfer Protocol. A standard protocol used to request and transmit information over the Internet or other computer network.

idle timeout

The amount of time a region or region entry may remain in the cache without being accessed before being expired. Access to an entry includes any get operation and any operation that resets the entry's time-to-live counter. Region access includes any operation that resets an entry idle timeout and any operation that resets the region's time-to-live.

Idle timeout attributes are set at the region level for the region and at the entry level for entries. See also time-to-live and expiration action.

initial capacity

Region attribute. The initial capacity of the map used for storing region entries.

invalid

The state of an object when the cache holding it does not have the current value of the object.

invalidate

Distributed: To mark an object as being invalid across the distributed cache.

Local: To mark an object as being invalid in the local cache only.

JDBC

Java DataBase Connectivity. A programming interface that lets Java applications access a database via the SQL language.

JMX

Java Management eXtensions. A set of specifications for dynamic application and network management in the J2EE development and application environment.

JNDI

Java Naming and Directory Interface. An interface to naming and directory services for Java applications. Applications can use JNDI to locate data sources, such as databases to use in global transactions. VMware GemFire allows its JNDI to be configured in a `cache.xml` configuration file.

JTA

Java Transaction API. The local Java interfaces between a transaction manager (JTS) and the parties involved in a global transaction. VMware GemFire can be a member of a JTA global transaction. See also global transaction.

JVM

Java Virtual Machine. A virtual machine capable of handling Java bytecode.

key constraint

Enforcing a specific entry key type. The key-constraint region attribute, when set, constrains the entries in the region to keys of the specified object type.

listener

An event handler. The listener registers its interest in one or more events, such as region entry updates, and is notified when the events occur.

load factor

Region attribute. The load factor of the map used for storing entries.

local

Local cache: The part of the distributed cache that is resident in the current member’s memory. This term is used to differentiate the cache where a specific operation is being performed from other caches in the same cluster or in another cluster. See also remote.

Region with local scope: A region whose scope is set to local. This type of region does not distribute anything with other members in the cluster.

Region shortcuts: In the RegionShortcut and settings, LOCAL means the scope is set to local. All client regions have local scope. In the ClientRegionShortcut settings, LOCAL means the region does not connect to the client’s servers.

local scope

Data distribution setting that keeps data private and visible only to threads running within the local member. A region with local scope is completely contained in the local cache. Client regions are automatically given local scope.

locator

VMware GemFire process that tracks system members and provides current membership information to joining members so they can establish communication. For server systems, the locator also tracks servers and server load and, when a client requests a server connection, the locator sends the client to one of the least loaded servers. .

LRU

Least recently used. Used to refer to region entry or entries most eligible for eviction due to lack of interest by client applications. VMware GemFire offers eviction controllers that use the LRU status of a region's entries to determine which to evict to free up space. Possible eviction actions are local destroy and overflow. See also resource manager.

machine

Any VMware GemFire-supported physical machine or Virtual Machine.

member

A process that has defined a connection to a VMware GemFire cluster and created a VMware GemFire cache. This can be a Java or Native Client application. This can also be a VMware GemFire process such as a locator or cacheserver. The minimal VMware GemFire process configuration is a single member that is connected to a cluster.

message queue

A first-in, first-out data structure in a VMware GemFire system member that stores messages for distribution in the same order that the original operations happened in the local member. Each thread has its own queue. Depending on the kind of queue, the messages could be going between two members of a cluster, a client and server, or two members in different clusters. See also conflation.

mirroring

See replicate.

multicast

A form of UDP communications where a datagram is sent to multiple processes in one network operation.

named region attributes

Region attributes that are stored in the member memory and can be retrieved through their region attributes refid setting. VMware GemFire provides standard predefined named region attributes, that are stored using region shortcut refids. You can use any stored attributes that you wish, setting an id when you create them and using the id setting in the refid you want to use to retrieve them.

netLoad

The method used by VMware GemFire to load an entry value into a distributed region. The netLoad operation invokes all remote cache loaders defined for the region until either the entry value is successfully loaded or all loaders have been tried.

netSearch

The method used by VMware GemFire to search remote caches for a data entry that is not found in the member's local cache region. This method operates only on distributed regions with a data-policy of empty, normal and preloaded.

netWrite

The method used by VMware GemFire to invoke a cache writer for region and region entry events. This method operates only on distributed regions. For each event, if any cache writer is defined for the region, the netWrite operation invokes exactly one of them.

network partitioning

A situation that arises from a communications partition that causes processes to become unaware of one another.

OQL

Object Query Language, SQL-92 extended for querying object data. VMware GemFire supports a subset of OQL.

off-heap memory

Memory that is not on the standard Java heap and that is not managed by the JVM and its garbage collector.

overflow

Eviction option for eviction controllers. This causes the values of LRU entries to be moved to disk when the region reaches capacity. Writes are performed according to the cache disk-store configuration.

oplog / operation log

The files in a disk-store used for the cache operations.

partition

The memory in each member that is reserved for a specific partitioned region's use.

partitioned region

A region that manages large volumes of data by partitioning it into manageable chunks and distributing it across multiple machines. Defining partition attributes or setting the region attribute data-policy to partition makes the region a partitioned region.

peer

A VMware GemFire member application that is not configured as a client. Peer configuration uses the <cache> `cache.xml` element and the Cache API. Peers can also be configured as servers to client applications and as gateway-receivers or gateway-senders to remote clusters.

persistent region

A region with the attribute data-policy set to persistent-replicate.

persistent-partition

A region attribute setting identifying a region as a partitioned region whose data is persisted to disk. With persistence, all region entry keys and values are stored in an operation log on disk as well as being stored in memory. Also referred to as disk region. Writes are performed according to the cache disk-store configuration.

persistent-replicate

A region attribute setting identifying a region as a replicate whose data is persisted to disk. With persistence, all region entry keys and values are stored in an operation log on disk as well as being stored in memory. Also referred to as disk region. Writes are performed according to the cache disk-store configuration.

producer

A VMware GemFire member process that puts data into the cache for consumption by other members. Producers may be configured with empty regions, where the data they put into the cache is not stored locally, but causes cache update events to be sent to other members. This is a common configuration in peer members and for client processes. See consumer.

pull model

Data distribution model where each process receives updates only for the data in which the process has explicitly expressed interest. In a VMware GemFire peer member, this is accomplished using a distributed, non-replicated region and creating the data entries that are of interest in the local region. When updates happen for the region in remote caches, the only updates that are forwarded to the local cache are those for entries that are already defined in the local cache. In a VMware GemFire client, you get pull behavior by specifically subscribing to the entries of interest. See push model.

push model

Data distribution model where each process receives updates for everything in the data set. In a VMware GemFire peer member, this is accomplished using a replicated region. All data modifications, creations, and deletes in remote caches are pushed to the replicated region. In a VMware GemFire client, you get push behavior by registering interest in all keys in the region. See pull model.

query string

A fully-formed SQL statement that can be passed to a query engine and executed against a data set. A query string may or may not contain a SELECT statement.

race condition

Anomalous behavior caused by the unexpected dependence on the relative timing of events. Race conditions often result from incorrect assumptions about possible ordering of events.

range-index

An XPath index optimized for range-queries with the added index maintenance expense of sorting the set of values. A range index allows faster retrieval of the set of nodes with values in a certain range. See also structure-index and value-index.

region

A logical grouping of data within a cache. Regions usually contain data entries (see entry). Each region has a set of region attributes governing activities such as expiration, distribution, data loading, events, and capacity control. In addition, a region can have an application-defined user attribute.

region attributes

The class of attributes governing the creation, distribution, and management of a region and its entries.

region data

All of the entries directly contained in the region.

region entry

See entry.

region shortcut

Enums RegionShortcut and ClientRegionShortcut defining the main region types in VMware GemFire for peers/servers and clients, respectively. Region shortcuts are predefined named region attributes.

remote

Resident or running in a cache other than the current member's cache, but connected to the current member's cache through VMware GemFire. For example, if a member does not have a data entry in the region in its local cache, it can do a netSearch in an attempt to retrieve the entry from the region in a remote cache within the same cluster. Or, if the member is a client, it can send a request to a server in an attempt to retrieve the entry from the region in a remote server cache in the server's cluster. In multi-site installations, a gateway sends events from the local cache to remote caches in other clusters. See also local.

replicated region

A region with data-policy set to replicate or persistent-replicate.

replicate

Region data-policy specification indicating to copy all distributed region data into the local cache at region creation time and to keep the local cache consistent with the distributed region data.

resource manager

VMware GemFire process that works with your JVM's tenured garbage collection (GC) to control heap use and protect your JVM from hangs and crashes due to memory overload. The manager prevents the cache from consuming too much memory by evicting old data and, if the collector is unable to keep up, by refusing additions to the cache until the collector has freed an adequate amount of memory. Eviction is done for regions configured for LRU eviction based on heap percentage. See also LRU and eviction-attributes.

rollback

A transactional operation that excludes a transaction's changes from the cache, leaving the cache undisturbed.

scope

Region attribute: In non-partitioned regions, a distribution property for data identifying whether it is distributed and, if so, whether distribution acknowledgements are required and whether distributed synchronization is required. A distributed region's cache loader and cache writer (defined in the local cache) can be invoked for operations originating in remote caches. A region that is not distributed has a local scope. See also replicate.

Querying: The data context for the part of the query currently under evaluation. The expressions in a SELECT statement's FROM clause can add to the data that is in scope in the query.

SELECT statement

A statement of the form SELECT projection_list FROM expressions WHERE expressions that can be passed to the query engine, parsed, and executed against data in the local cache.

serialization

The process of converting an object or object graph to a stream of bytes.

server

A VMware GemFire member application that is configured as a peer in its own system and as a server to connecting VMware GemFire client applications.

server group

An optional logical grouping of servers in a server clusters. There is always the default server group made up of all available server in the server clusters. Clients can specify the server group in their server pool configuration. Then the pool only connects to those servers. If no group is specified, the default is used.

server connection pool

The cache entity that manages client connections to servers.

socket

The application interface for TCP/IP communications. UDP provides unicast and multicast datagram sockets, while TCP provides server and connection sockets. TCP server sockets are used by server processes to create connection sockets between the server and a client.

SQL

Structured Query Language.

SSL

Secure Socket Layer. A protocol for secure communication between Java VMs.

standalone distributed system

A cluster configured for no communication with peers. Client applications are generally defined with standalone clusters, so there is no peer communication and all event and data communication is done between the client member and the server tier.

statistics enabled

Region attribute. Specifies whether to collect statistics for the region.

struct

A data type that has a fixed number of elements, each of which has a field name and can contain an object value.

structure-index

An XPath index that is basically a pre-computed query. Any legal XPath expression can be used. The index maintains lists of all nodes that match the expression used to create it. If a query is performed that has the same expression as the index then the result is available without XPath evaluation. See also range-index and value-index.

system member

See member.

TCP

The Transmission Control Protocol is a part of the internet protocol (IP) suite that provides unicast communications with guaranteed delivery. The TCP protocol is connection-based, meaning that a TCP socket can only be used to send messages between one pair of processes at a time. Compare to UDP.

timeout

A behavior associated with synchronization functions. Timeout behavior is exhibited as refusal to wait longer than a specified time for a signal to proceed. See also blocking.

time-to-live

The amount of time a region or region entry may remain in the cache without being modified before being expired. Entry modification includes creation, update, and removal. Region modification includes creation, update, or removal of the region or of any of its entries.

Time-to-live attributes are set at the region level for the region and at the entry level for entries. See also idle timeout and expiration action.

transaction

See cache transaction and global transaction.

transaction listener

User-implemented plug-in for receiving and handling transaction events. A transaction listener is called after a transaction commits. See also transaction writer.

transaction writer

User-implemented plug-in intended for synchronizing the cache with an outside data source. A transaction writer is a synchronous listener to cache transactions. The transaction writer has the ability to veto a transaction. See also transaction listener.

transactional view

The result of a history of transactional operations for a given open transaction.

transport layer

The network used to connect the VMware GemFire system members in a VMware GemFire system.

TTL

See time-to-live.

UDP

The User Datagram Protocol is a part of the internet protocol (IP) suite that provides simple, unreliable transmission of datagram messages from one process to another. Reliability must be implemented by applications using UDP. The UDP protocol is connectionless, meaning that the same UDP socket can be used to send or receive messages to or from more than one process. Compare to TCP.

unicast

A message sent from one process to another process (point-to-point communications). Both UDP and TCP provide unicast messaging.

URI

Uniform Resource Identifier. A unique identifier for abstract or physical resources on the World Wide Web.

user attribute

An optional object associated with a region or a data entry where an application can store data about the region or entry. The data is accessed by the application only. VMware GemFire does not use these attributes. Compare to region attributes, which are used by VMware GemFire.

value constraint

Enforcing a specific entry value type. The value-constraint region attribute, when set, constrains the entries in the region to values of the specified object type. Value constraints can be used to provide object typing for region querying and indexing. The value-constraint is only checked in the cache that does the entry put or create operation. When the entry is distributed to other caches, the value constraint is not checked.

value-index

An XPath index that operates much as a structure-index does, but that separates the nodes that match the XPath expression into sets mapped by each node’s value. This allows further filtering of the nodes to be evaluated in a query by going directly to those with a specific value. See also structure-index and range-index.

view

A collection of member identifiers that defines the membership group.

Virtual Machine

A completely isolated operating system installation within your normal operating system. This is generally implemented by software emulation or hardware virtualization.

VMware virtual machine

Also referred to as a VMware VM. A VMware VM is a tightly isolated software container that can run its own operating systems and applications as if it were a physical computer. A VMware VM behaves exactly like a physical computer and contains it own virtual (ie, software-based) CPU, RAM hard disk and network interface card (NIC). An operating system can't tell the difference between a VMware VM and a physical machine, nor can applications or other computers on a network. Even the VMware VM thinks it is a “real” computer. Nevertheless, a VMware VM is composed entirely of software and contains no hardware components whatsoever.

XML

EXtensible Markup Language. An open standard for describing data, XML is a markup language similar to HTML. Both are designed to describe and transform data, but where HTML uses predefined tags, XML allows tags to be defined inside the XML document itself. Thus, virtually any data item can be identified. The XML programmer creates and implements data-appropriate tags whose syntax is defined in a DTD file or an XSD (XML schema definition.)

XML schema definition

The definition of the structure, content, and semantics used in an XML document. The definition can be used to verify that each item of content in a document adheres to the specification of the element in which the content is placed. The XML schema is a superset of DTD. Unlike DTD, XML schemas are written in XML syntax, which, although more verbose than DTD, are more descriptive and can have stronger typing. Files containing XML schema definitions generally have the XSD extension.

XPath

A language that describes a way to locate and process items in Extensible Markup Language (XML) documents by using an addressing syntax based on a path through the document's logical structure or hierarchy.

XSD

See XML schema definition.