Abstract

This administrators guide describes administration of the EMC Real-Time Intelligence (RTI) platform for Big and Fast Data. This includes installation, deployment, configuration, monitoring, and customization of RTI.

302-003-696

REV 02
## Contents

### Chapter 1  Administrating the EMC RTI System  
Administrating the EMC RTI System ................................................................. 9
Identifying the administrator’s role ................................................................. 10
We value your feedback! .................................................................................. 10

### Chapter 2  Installing RTI Software  
Installing the EMC RTI Software .................................................................. 11
System requirements ...................................................................................... 12
Adding a server host to RTI ........................................................................... 12
Security requirements .................................................................................... 13
Disk space requirements .............................................................................. 14
Installing RTI and prerequisites .................................................................... 15
Troubleshooting the installation ..................................................................... 16
Licensing RTI ................................................................................................. 17

### Chapter 3  Configuring an RTI System  
Configuring an RTI system .......................................................................... 18
Getting started with RTISH .......................................................................... 19
Creating and enabling applications ............................................................... 21
Setting properties for applications ............................................................... 22
Configuring a single-host system ................................................................. 25
Configuring a multi-host system ................................................................. 30
Configuring output streams ......................................................................... 38
Filtering output streams ............................................................................. 40
Maximum Java virtual machine size ............................................................ 43
Using cascading hash ................................................................................... 44

### Chapter 4  Loading Reference Data  
Loading reference data ................................................................................ 45
RDL error messages ...................................................................................... 61

### Chapter 5  Controlling Event Flows  
Controlling event flow ................................................................................ 64
Event flows through the distribution grid ...................................................... 65
Location-based service provisioning ............................................................ 69

### Chapter 6  Subscribing to Events  
Subscribing to events ................................................................................... 74
Provisioning data model ............................................................................. 75
## Contents

Anonymizing data ........................................................................................................ 76
Setting up a new event subscription ........................................................................ 77

**Chapter 7  Enhancing Event Data**  88
Ingesting events .......................................................................................................... 89
Enriching data ............................................................................................................. 90
Using selectable geocoding ....................................................................................... 93
Tracking geofence activity .......................................................................................... 97

**Chapter 8  Using Metrics and Analyzing Events**  102
Using metrics and analyzing events .......................................................................... 103
Managing event analysis ............................................................................................ 103
Managing metrics ........................................................................................................ 112
Creating user-defined metrics streams ..................................................................... 133
KPI store function ....................................................................................................... 137

**Chapter 9  Exporting Data**  138
Exporting data ............................................................................................................ 139
Using AMQP exchange .............................................................................................. 139
Using a CSV file ......................................................................................................... 142
Using a JDBC data source ......................................................................................... 143
Using HDFS transport to export events into Hadoop ................................................ 144
InfluxDB transport ...................................................................................................... 155

**Chapter 10  Managing the RTI System**  157
Managing the RTI System .......................................................................................... 158
Monitoring IG to DG connectivity and flow ............................................................... 166
Restarting the system ................................................................................................. 168
Modifying configuration properties .......................................................................... 172
Backup and restore guidelines .................................................................................. 172
Recovering from a system failure .............................................................................. 179
RTI error messages ..................................................................................................... 182
RTI log files ................................................................................................................ 185

**Chapter 11  Using the System Tools**  189
Using the system tools ............................................................................................... 190
Validation mode for analytics .................................................................................... 192
Sample output report ................................................................................................. 192
Syphon utility syntax .................................................................................................. 192
Using GemFire shell ................................................................................................... 196
Using the REST interface to inspect data ................................................................. 196
Using the HDFS Transport inspection tool .............................................................. 201
Contents

Using RTISH ................................................................. 202
Reference data loader (RDL) ............................................. 203

Chapter 12  Virtualizing RTI Infrastructure 204
RTI on virtualized environments best practices .................. 205

Chapter 13  Monitoring and Tuning Event Throughput 207
Monitoring and tuning event throughput .......................... 208
Tuning RTI and the supporting infrastructure ..................... 208
Scaling and monitoring over time ....................................... 210
Understanding horizontal and vertical scaling in RTI .......... 210
Scaling to increases in throughput ..................................... 210
Event throughput overload on RTI applications .................. 211
Tuning flow control and parallelization ............................ 215
Tuning the ingester ......................................................... 221
Tuning the ingest grid ...................................................... 224
Tuning the distribution grid .............................................. 229
Tuning the number of buckets to rebalance partitions .......... 233
Stateful analytics and metrics ........................................... 234
Output stream filtering .................................................... 234
Setting event threshold ................................................... 235

Chapter 14  Configuration Properties 236
Configuration properties .................................................. 237
Common properties ....................................................... 237
Protocol adapter properties ............................................. 240
GemFire cache server properties ...................................... 242
GemFire pool properties .................................................. 243
GemFire disk store properties .......................................... 244
Ingester properties ......................................................... 245
Distribution grid properties ............................................. 250
Provisioning Manager and RabbitMQ properties ............... 265
Event format properties .................................................. 267
RDL properties ............................................................. 270
REST API client properties ............................................. 271
HDFS properties ........................................................... 271

Chapter 15  Referencing RDL Commands 272
RDL commands ............................................................ 273
buildSpatialIndex command ............................................ 273
create command ............................................................ 274
delete command ........................................................... 275
fetchSubscriberMapping command ................................................................. 276
rebuildOptins command ............................................................................. 276
subscriptionState command ...................................................................... 277
update command ......................................................................................... 277

**Chapter 16  RTISH Commands**  279

RTISH commands ......................................................................................... 282
Basic RTISH operations ............................................................................... 282
rti create application .................................................................................. 283
rti declare exchange .................................................................................. 283
rti declare queue ......................................................................................... 284
rti delete exchange .................................................................................... 284
rti delete queue ......................................................................................... 284
rti deploy host ............................................................................................ 285
rti disable ingester protocol ....................................................................... 285
rti disable instance .................................................................................... 285
rti disable ssh-key ...................................................................................... 285
rti enable ingester protocol ....................................................................... 286
rti enable instance ...................................................................................... 286
rti enable ssh-key ...................................................................................... 286
rti ext install ............................................................................................... 286
rti jmx invoke ............................................................................................. 287
rti license audit ........................................................................................... 287
rti license deploy ......................................................................................... 288
rti list application types ............................................................................. 289
rti list env .................................................................................................... 289
rti list hosts ................................................................................................. 290
rti list instances .......................................................................................... 290
rti list mbeans ............................................................................................. 290
rti list processes .......................................................................................... 290
rti list properties .......................................................................................... 291
rti list ssh-keys ........................................................................................... 291
rti logs collect .............................................................................................. 291
rti logs list ..................................................................................................... 293
rti logs show ................................................................................................ 293
rti metrics deploy ....................................................................................... 294
rti metrics describe .................................................................................... 294
rti metrics disable ........................................................................................ 295
rti metrics enable ........................................................................................ 295
rti metrics list ............................................................................................... 296
# Contents

**Chapter 17  Reference Data Formats**
Reference data formats .................................................. 311

**Chapter 18  Loading Reference Data with RTISH**
Loading reference data with RTISH .................................. 328
upsert command .............................................................. 329
delete command .............................................................. 330
Command line arguments .................................................. 330
Operational commands ..................................................... 331
Tuning options ............................................................... 332

**Chapter 19  Provisioning Manager Message Interface for Loading Reference Data**
Overview ................................................................. 334
Common message elements .............................................. 335

**Chapter 20  Output Format for Event Streams**
Output format for event streams ....................................... 348

**Chapter 21  Setting up a Grafana Dashboard**
Setting up a Grafana dashboard ....................................... 356

**Appendix A  Glossary**
Glossary ............................................................................ 358
Chapter 1   Administrating the EMC RTI System

This chapter presents the following topics:

Identifying the administrator’s role

We value your feedback!
Chapter 1: Administrating the EMC RTI System

Identifying the administrator’s role

In EMC Real-Time Intelligence (RTI), the Administrator loads data, issues command-line instructions, configures the system, and uses tools to monitor, troubleshoot, and tune the system.

RTI also uses Pivotal GemFire capabilities. For information about GemFire, refer to the GemFire documentation.

Administrative tools

An Administrator uses these tools to administer RTI:

- The Syphon Utility
- GemFire Shell (GFSH)
- REST Interface to Inspect Data
- HDFS Transport Inspection Tool
- RTISH (RTI shell tool)
- Reference Data Loader (RDL)

We value your feedback!

The authors of this document welcome your feedback on the solution and the solution documentation. Contact EMC.Solution.Feedback@emc.com with your comments.

Authors: Lyndon Adams, Vijayanand Bharadwaj, Brice Bingman, Charlie Black, Simon Gammage, Damu (Tom) Liu, Gideon Low, Esteban Serrano, Reed Tucker, Qing Ye
Chapter 2  Installing RTI Software

This chapter presents the following topics:

- Installing the EMC RTI Software .......................................................... 12
- System requirements ............................................................................. 12
- Adding new server host to RTI.............................................................. 12
- Security requirements .......................................................................... 13
- Disk space requirements ....................................................................... 14
- Installing EMC RTI and prerequisites .................................................. 15
- Troubleshooting the installation .............................................................. 16
- RTI Licensing ......................................................................................... 17
Chapter 2: Installing RTI Software

Installing the EMC RTI Software

This section explains how to install the RTI software on an initial target system. For information about deploying the RTI software to additional nodes and configuring the components that make up a complete RTI system, see Configuring an RTI system.

This section also identifies system requirements and prerequisite third-party software that must be installed for the RTI system to run.

System requirements

The following table describes the system requirements for installing and running RTI.

Table 1. System requirements

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Description</th>
</tr>
</thead>
</table>
| Operating system | Red Hat Enterprise Linux (RHEL) (x86_64) or CentOS Version 6.4 or 7.0 (x86_64)  
See [https://access.redhat.com/site/downloads/](https://access.redhat.com/site/downloads/)  
SSH must be enabled and configured for public key authentication on all nodes.  
TCP SYN Cookies must be disabled on all target deployment hosts. On RHEL this means updating /etc/sysctl.conf to include:  
net.ipv4.tcp_syncookies=0  
Transparent Huge Pages (THP): We recommend that this Linux feature be disabled on hosts running RTI to avoid excessive memory fragmentation and significantly increased CMS Garbage Collection (GC) overhead. |
| Java | Oracle JDK Version 7u75 or later (64-bit)  
Oracle JDK Version 8u91 or later (64-bit)  
| RabbitMQ | Version 3.1.x  
See [http://www.rabbitmq.com/download.html](http://www.rabbitmq.com/download.html) |

The macOS operating system is also supported for test and development purposes.

Adding a server host to RTI

1. Update the RTI runtime user's JAVA_HOME environment variable (in .bash_profile and .bashrc) to point to the configured Java runtime bin directory.
2. Enable password-less SSH between the new host and all existing RTI hosts (in both directions).
3. Disable Linux Firewall/SE Linux features. This makes RTISH run much faster as remote SSH connections establish themselves more quickly. Note that RTISH validates the consistency of the cluster configuration across all deployed hosts each time it launches, and copies changes to all deployed hosts each time a
change is made. Therefore, fast SSH handshakes have the largest impact on
routine-task RTISH performance.

4. Make sure to deploy RTI to a directory tree rooted to a disk with plenty of space
for both logging and output event record files. If this is difficult to accomplish for
some reason, make sure the map RTI's output directories to a disk partition with
plenty of space. See: Disk space requirements.

5. Make sure that TCP SYN Cookies are disabled. Failure to do so will result in an
unstable system.

6. Make sure that the /etc/hosts file is identical for all hosts running RTI instances,
and on any other hosts from which you plan to run RTISH.

7. Make sure that the maximum processes environment variable ("nproc") is set to at
least 4096. The default Linux value of 1024 is often not high enough for RTI.

8. Make sure that the JMX port (1099) is open between your management
workstation and the new RTI server so you can monitor RTI using JMX. You can
choose a different dedicated port if necessary.

9. For any other OS-level changes you may have applied to your other deployed RTI
hosts, be sure to make the same changes on the new host. RTI is designed to
run on a homogenous server farm. Different hosts intended to run different RTI
applications may have different hardware-level configurations (CPU's, memory,
and so on), but the OS settings should be the same.

10. We recommend the latest version of Java 8 as the default JRE for RTI 2.3.0 and
later. We also recommend that you install the full JDK to enable RTISH process-
attach features. Java 7 is still supported as of RTI 3.0, but provides substantially
worse Garbage Collection performance. If you are adding a new host to your RTI
cluster, this might be a great opportunity to upgrade all your hosts to Java 8. Note
that all RTI instances must always use the same version of Java.

Security requirements

Before installing the RTI software, read the following notes about securing the system.

The ingester and the ingest grid are typically installed into a fire-walled Network
Operations Center (NOC) where the network probe resides. Users who need access to
RTI components inside the NOC must be subject to strict security authorization consistent
with security measures in place for other NOC operations. In general, a NOC is a trusted,
secure facility, and RTI does not need to provide any security mechanisms to further
protect this environment.

Communication between the ingest grid and the distribution grid is one-directional. Data
may not flow from the distribution grid to the ingest grid. The GemFire Gateway is a WAN
gateway that connects the grids, using a TCP/IP connection over a secure VPN. In
general, the distribution grid should be installed in a separate data center with no
connectivity into the NOC.

Most of the reference data loaded into RTI is loaded into the ingest grid and propagated to
the distribution grid. Access to certain RDL commands may need to be restricted to
protect the system from inadvertent data updates, refreshes, and deletes. Administrators
may need to run all RDL commands and load all data types. Regular operators may be restricted to loading certain data types and running a subset of the commands.

The Provisioning Manager and RabbitMQ components of RTI are end user interfaces that require access to the distribution grid for the RTI system to work; therefore, the data center where the distribution grid resides is a less secure environment. RabbitMQ and Provisioning Manager user privileges need to be carefully defined to protect the RTI system.

### Disk space requirements

Although RTI is a system that processes event streams in memory, you need to take the following writeable directories into account when you configure disks for RTI hosts:

- **/var/log/<instance_name>** — RTI log files and GemFire statistics files. EMC recommends allocating at least 50GB of space for the logs. To free up space, you can set the maximum amount of disk space that can be used before old logs are deleted. See RTI log files.

- **/var/data/<instance_name>** — PID files (for all RTI processes). RTISH uses these files to check the health of processes or to stop them. This directory has no appreciable space requirements.

- **/var/data/<grid>/diskstore** — GemFire disk store files for the two grids. The disk stores are used for two purposes: recovery of reference data after system restarts and overflow-to-disk space when a server starts to run low on memory. EMC recommends sizing the system such that all data fits comfortably in memory. This approach implies that disk space usage for disk stores should not be much greater than the expected memory usage for the data stored in cache. However, to err on the safe side, EMC recommends allocating at least twice as much disk space for disk stores as memory required for the RTI server processes. If expected reference data sets exceed 655GB (per RTI installation), adjust the rti.gfe.disk-store.max-size property on the grids. See GemFire disk store properties.

- **/var/data/<distgrid_instance_name>/location** and **/var/data/<distgrid_instance_name>/transaction** — Transaction and location event output files, which are available for external consumption. Depending on how often these files are consumed and then deleted (by processes outside RTI), the space occupied by these files may increase fairly quickly. EMC recommends a minimum of 1TB of available disk space; however, you may choose to allocate less space for this purpose, depending on your external procedures for processing and deleting the files.

For more information, refer to Chapter 10 Managing the RTI System.
Installing RTI and prerequisites

Follow these steps in order.

1. Plan your implementation of RTI in terms of the physical servers you will use. Single-node and multi-node configurations are supported. See Chapter 3 Configuring an RTI System for examples.

2. Enable and configure SSH on all servers for public key authentication:
   a. Create a public/private key:
      ```bash
      bash$ ssh-keygen -t rsa -C "yourname@yourdomain.ext"
      ```
   b. Allow public/private key authentication on your local machine:
      ```bash
      bash$ cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
      ```
   c. Allow public/private key authentication on remote machines:
      ```bash
      bash$ cat ~/.ssh/id_rsa.pub | ssh user@remote.machine.com 'cat >> ~/.ssh/authorized_keys'
      ```

3. Install the Java Development Kit (JDK) on the EMC RTI nodes.

4. Make sure that the JAVA_HOME variable is set and points to the correct JDK installation:
   ```bash
   bash$ env|egrep JAVA_HOME
   JAVA_HOME=/opt/jdk/jdk1.7.0_45/
   ```

5. Install RabbitMQ 3.1.x on a system that will have connectivity to the distribution grid and the Provisioning Manager. In a multi-node system, RabbitMQ is typically installed on its own node or shares a node with the Provisioning Manager.
   a. See http://www.rabbitmq.com/download.html
   b. See also "RabbitMQ 3.1 Supported Configurations and System Requirements": http://pubs.vmware.com/vfabricRabbitMQ31/topic/com.vmware.ICbase/PDF/vfabric-rabbitmq-suppconfigs-3.1.5.pdf

6. Copy the EMC RTI tarball to the initial target server.

7. Extract the tar file in your base directory. For example:
   ```bash
   bash$ sudo tar xzf rti-t2.1.3.RELEASE-full.tar.gz -C /opt/
   ```

The resulting `emc_rti_<version>` directory contains all of the RTI software required to start configuring the system. See Chapter 3 Configuring an RTI System for instructions on deploying and setting up the system components.
Troubleshooting the installation

java.lang.OutOfMemoryError: unable to create new native thread

The default Red Hat Enterprise Linux (RHEL) Max User Processes (nproc) value of 1024 is low. Exceeding this limit causes RTI to crash, producing the error. To prevent this situation, increase the default value for nproc in /etc/security/limits.conf.

For example:

bash$ vim /etc/security/limits.conf

*      hard    nproc    8192
njones soft    nproc    4096
ksmith soft    nproc    4096

This setup defaults the hard limit to 8192 for all users and sets a soft limit of 4096 for two specific users.

EMC does not recommend setting the nproc values with the ulimit -u option, because this setting can cause unpredictable behavior. Each thread spawned by a user across all sessions is counted towards the maximum limit, so this setting is a global user-level maximum value. The ulimit -u option increases the limit for the current session only, even though new threads spawned by the current session count toward the global process count. Therefore, other applications or sessions could be impacted.

Depending on system configuration, RHEL/CentOS may require that the values be updated in the /etc/security/limits.d/90-nproc.conf file.

Confirming configuration details at the process level

When debugging system behavior, it can be challenging to confirm that a process has picked-up all of the expected configuration details. If you have doubts whether a process is configured as expected, review the head of the RTI log file. The file contains output of the complete RTI and Java level properties configuration, as well as a listing of all default values (that are being used because they have not been explicitly overridden). It is important to provide this output to RTI Support any time you open a support request.

Note: RTI configuration properties and versioning details are printed at the head of each RTI application log file. The output is broken-up into the following sections:

- **Java System Properties**: Applied via Java “-D” startup parameters, and usually configured via the RTI_OPTS environment variable (see RTISH command “rti set env”).
- **RTI Properties Defined with RTISH**: These are the properties explicitly configured by RTI end-users (see RTISH command “rti set property”).
- **RTI Properties retaining default values**: These are properties that have not been changed by RTI end-users, retaining their default values.
Chapter 2: Installing RTI Software

- **RTI Environment Variables:** There are several of these that define RTI’s local operating environment. Typically only `RTI_OPTS` is assigned by end-users.
- **Java “jar” File Archives:** A listing of all .jar files found in the Java classpath.

### Licensing RTI

To run RTI, install a valid license issued by EMC. Contact your RTI representative to obtain the license. Details about the RTI licensing model and different types of licenses are available from your RTI representative as well. The scope of this guide is limited to describing the steps to deploy and audit license files.

#### License format and contents

RTI license files have a `.lic` extension. A license file can contain one or more “license keys” or “feature lines.” Each line begins with the keyword `INCREMENT` followed by various attributes. RTI has only one licensed feature known as `RTI_BASIC`.

Always run RTI with a valid license file. “Valid” implies that the file has a `.lic` extension, licensed features have not expired, the file is correctly formatted, and has not been altered in any way.

#### Installing licenses

You must install license files in the folder `rti-t` with the rest of the RTI artifacts. This location is fixed. This folder is found under the `emc_rti_<version>` folder when the RTI binary tar file is unpacked. Utilities that deploy and audit licenses will always look for license files under the `rti-t` folder only.

You must install the license file on every host of the RTI cluster. You can deploy the file using the `rti license deploy` command on all hosts of the RTI cluster after you have deployed your cluster. The command deploys one file at a time. Refer to the sections Configuring a single-host system and Configuring a multi-host system in Chapter 3 Configuring an RTI System for steps to install the license file using the command. Refer to Chapter 16 RTISH Commands for the command syntax.

You can also manually copy the license files to each host, but we recommend using the command since it takes care of installing the file in the correct location, and it detects errors during installation. If needed, multiple license files can be installed. You might need to install more than one file when upgrading usage capabilities (extend time period or CPU cores) for the feature. In such a case a new license file is provided and you can install it with the existing file.

#### License auditing

The `rti license audit` command audits all license files present on the same host on which it is run. This command reports details of each licensed feature including type, expiration, core count etc. Refer to Chapter 16 RTISH Commands for the command details.

#### Removing invalid licenses

You must manually remove files with expired features, incorrectly formatted features, and altered files from all hosts of your RTI deployment.
This chapter presents the following topics:

- Configuring an EMC RTI system .......................................................... 19
- Getting started with RTISH ................................................................. 19
- Creating and enabling applications ...................................................... 21
- Setting properties for applications ...................................................... 22
- Configuring a single-host system ......................................................... 25
- Configuring a multi-host system .......................................................... 30
- Configuring output streams ................................................................. 38
- Filtering output streams ....................................................................... 40
- Maximum Java virtual machine size ...................................................... 43
- Using cascading hash .......................................................................... 44
Configuring an RTI system

After you have copied the RTI installation package (a tarball) to a Linux host and extracted its contents, you can use the RTISH tool to create system components known as applications, enable them on specific hosts, and start the system.

An application enables you to manipulate a set of instances as a single entity. An instance in this context refers to a particular application on a particular host. For example, an instance of the application myapplication on server svr0001 is known as instance myapplication@svr0001. You can create an application by using the rti create application command and specify where the application should run by using rti enable instance.

You can manipulate applications and instances by using a wildcard notation:

- You can refer to all instances of an application: myapplication@
- You can refer to all instances of all applications on a particular host by replacing the application name with an asterisk: *@svr0001
- You can refer to all instances in an RTI installation by replacing both the application name and the host name with asterisks: *@

The instances for an application do not all have to be configured in exactly the same way. If necessary, you can set configuration properties or environment variables for a single instance.

RTI supports two deployment models:

- **Single-host**: All applications and the RTI shell reside on a single node. RTISH launches on that host and manages the locally configured applications.
- **Multi-host (a distributed RTI cluster)**: RTI is installed on a management node first. RTISH launches on that host but is used to configure and manage applications remotely.

The sections below describe in detail the steps used to deploy RTI in the above modes with examples. We also provide sample scripts with configuration settings for both single and multi-host deployments. These scripts should be customized for your environment. These files are in the rti-t/samples/deployment-scripts folder of the installation.

Getting started with RTISH

RTISH is a command-line interface (CLI) that you can use to install, configure, and manage components in the RTI system. You can work with components on the machine where RTISH is installed and on remote hosts. To start RTISH:

```bash
$ cd <rti root>
$ ./common/rtish/bin/rtish
...
rtish>
```
You can also start RTISH and run a script directly from the OS shell. For example:

% path/to/rtish --cmdfile example.cmd

RTISH is interactive and supports tab-completion. The help command lists all of the available commands. You can also display help for individual commands. The following commands will familiarize you with the RTISH interface:

<table>
<thead>
<tr>
<th>Task</th>
<th>RTISH command</th>
</tr>
</thead>
<tbody>
<tr>
<td>Show a list of available commands</td>
<td>rtish&gt; help</td>
</tr>
<tr>
<td>Show help for an RTISH command</td>
<td>rtish&gt; help rti create application</td>
</tr>
<tr>
<td>Show RTISH version</td>
<td>rtish&gt; version</td>
</tr>
<tr>
<td>Show RTISH properties</td>
<td>rtish&gt; system properties</td>
</tr>
<tr>
<td>Run a script from RTISH</td>
<td>rtish&gt; script --file example.cmd</td>
</tr>
<tr>
<td>Exit from RTISH</td>
<td>rtish&gt; exit</td>
</tr>
</tbody>
</table>

The first time you execute RTISH on a host, the tool detects that the host contains a new RTI installation. Everything you do to configure the new system is represented in an XML configuration file:

`<install_dir>/emc_rti_<version>/rti-t/common/rtish/etc/cluster-config.xml`

When you deploy RTI to additional hosts, this file is copied and maintained on those hosts. Before creating any applications, check the name of your local host (where you are running RTISH):

```
rtish> rti list hosts
```

Assuming that RTI is freshly installed, the output should be a single line. For example:

`192.168.0.100`

OR

`srv001.example.org`

When you run RTISH for the first time, it might fail to find a globally visible name for the current host. In this case, the system chooses a name that only makes sense in the local context, such as `localhost`, and returns a warning message. To make the host visible from different machines, use the `rti rename host` command to assign a globally visible name, such as an IP address or a DNS host name, to the host.
Creating and enabling applications

Regardless of the hardware topology of your RTI implementation, you must complete the following tasks:

1. Create applications of the following type:
   - Locator
   - Ingest grid

   **Note:** Since version 3.1.0, the Ingester and ingest grid applications have been combined for improved performance. Therefore it is no longer necessary to create a separate Ingester instance.

   - Distribution grid
   - Provisioning Manager
   - REST server

2. Enable instances of these applications. When an application has been created, it must be enabled. All hosts can be configured with the complete set of applications you create, but you control which hosts have active instances of a given application.

3. Set properties for application instances. This step completes the configuration by assigning connectivity settings and other general properties.

4. Start the instances, in order.

The first RTI application required in a deployment is a locator, which defines the hostname and port for the hosts where the RTI system is going to run. Create a locator application by running an `rti create application` command. Locator processes coordinate and manage the grids as a directory of nodes.

```
rti create application --type locator --name basic-locator-1
```

After creating the application, set its port and hostname properties. The name consists of the locator name, followed by the `@` symbol and the host name (basic-locator-1@localhost in this example).

Locators are required for everything except RTISH. They are the "well-known addresses" that bind together the deployed clusters (and allow clients to discover the servers in the clusters). Each system component must connect to the GemFire Locators that are configured.

```
rti set property --instances basic-locator-1@localhost --name rti.locator.port --value 10334

rti set property --instances basic-locator-1@localhost --name rti.locator.host --value 127.0.0.1
```
After creating an application and setting its properties, enable an instance of the application on a specific host (or all hosts). Instances must be enabled before they can be started. This approach makes it possible to run different components on different machines. The following command enables an instance of the locator application on the local host.

```
rti enable instance --instances basic-locator-1@localhost
```

Start the locator instance with this command:

```
rti start --instances basic-locator-1@localhost
```

To determine if the start command has started the service, run this command:

```
rti status
```

To stop all running services, run this command:

```
rti system shutdown
```

After creating the locator instances, create applications for the following RTI components:

- Ingest grid (application type ingestgrid)
- Distribution grid (application type distgrid)
- Provisioning Manager (application type provisioning)

Like the locator instances, you can enable instances for these applications selectively on one or more hosts.

The following command displays a list of all enabled application instances:

```
rtish> rti list instances
```

### Setting properties for applications

Depending on the type of application, you need to set appropriate properties that further define the application and its environment. For example, you can set filter properties for the Ingester. Some properties are common to all components, such as connectivity properties. For a list of the properties that are available for each RTI component, see Configuration properties. For specific examples, see Configuring an RTI system.

### Starting and stopping instances

After creating instances, setting properties for instances, and enabling instances, you can start them. To start components in the right order, starting with the locators, use the `rti system start` command. See Configuring an RTI system for specific examples.

### Configuring ports

A critical step in configuring an RTI system is the assignment of ports to support connectivity among the server components and clients. The minimum port requirements support GemFire connectivity, JMX monitoring, and the RabbitMQ service.
GemFire has separate and distinct connectivity layers for peer-to-peer, client/server, and WAN replication purposes. However, these layers share common GemFire locator processes for dynamic discovery of all non-locator ports. GemFire ports fall into two categories:

- Ports that must be explicitly configured (or at least a default must be set)
- Ports that may be dynamically assigned (although you can specify a port range for these dynamic assignments)

For more information about GemFire port usage and associated firewall configuration requirements, see the GemFire documentation.

RabbitMQ must be installed separately from RTI. See the RabbitMQ documentation for details on how to configure a custom RabbitMQ listener port (default 5672). Distribution grid and Provisioning Manager application instances automatically try to connect to the default port if none is specified by the `rti.rabbit.addresses` property.

For more information about setting RTI application properties, see Configuration properties.

These ports are non-dynamic ports that administrators must configure for the RTI system (or accept the default values).

**Ports for GemFire locators**

Locator ports default to 10334. These ports must be accessible to all RTI GemFire servers in a cluster, and to all RTI GemFire client applications, including Ingesters, the Provisioning Manager (PM), the Reference Data Loader (RDL), REST server, and RTISH.

- The PM only needs to access the distribution grid locator ports.
- RTISH and RDL both need to access the locator ports on the locally managed cluster.
- Locator processes themselves, as well as ingest grid servers, must have visibility to the locator ports on both the ingest grid and the distribution grid (otherwise, WAN replication will not work).

You configure the locator port for an application with the following RTISH command:

```
rti set property --instances <InstanceName>@<Hostname|*> --name rti.locator.port --value 10334
```

**Ports for JMX monitoring**

The standard value for the main JMX Manager port is 1099. (The RTI property name is `rti.gemfire.jmx-manager-port`.) This port is exposed by each long-lived RTI process (ingest grid and distribution grid servers, Ingesters, and the Provisioning Manager). The port must be accessible to whatever JMX client you choose to use. EMC also recommends access for RTISH as well, to enable its system introspection features.

The standard value for the JMX Manager HTTP port is 8080. (The RTI property name is `rti.gemfire.jmx-manager-http-port`.) This port is used to connect the GemFire Pulse monitoring web application.
Configure these ports with the following RTISH commands:

\[
\begin{align*}
\text{rti set property --instances } & \langle\text{InstanceName}\rangle@\langle\text{Hostname}|*\rangle \ --\name \rti.gemfire.jmx-manager-port \ --\value \ 1099 \\
\text{rti set property --instances } & \langle\text{InstanceName}\rangle@\langle\text{Hostname}|*\rangle \ --\name \rti.gemfire.jmx-manager-http-port \ --\value \ 8080
\end{align*}
\]

**Membership port range**

GemFire provides a mechanism for specifying port ranges. The default range is 1024–65535. Ports in this range are used for UDP Unicast Gossip Protocol messaging, FD.SOCK cluster health heartbeat monitoring, TCP communications for peer-to-peer data replication, and any other GemFire peer-to-peer traffic.

The range must specify a minimum of three ports to support the three different protocols; however, EMC recommends at least 30 ports to ensure that certain failure and recovery scenarios do not delay restart times.

The entire range must be visible only to members of the local server cluster (either the ingest grid or the distribution grid, inclusive of each grid’s locator processes). No clients will attempt to reach these ports.

Configure the range with the following RTISH command:

\[
\begin{align*}
\text{rti set property --instances } & \langle\text{InstanceName}\rangle@\langle\text{Hostname}|*\rangle \ --\name \rti.gemfire.membership-port-range \ --\value \ 54000-54050
\end{align*}
\]

**Protocol adapter listener ports**

The ingest grid application type listens for input event streams via TCP listener ports. These are the default values for several protocol adapter options:

- adr -- 29001
- gb -- 29002
- iucs -- 29003
- iups -- 29004
- s1mme -- 29006
- gngi -- 29007

**Dynamic ports**

Dynamic ports are ports that the system dynamically assigns. These dynamic assignments may be based on the default membership port range or a range that you specify with the rti.gemfire.membership-port-range property. These ports are defined per application instance, and you need the same number of defined ports regardless of the scale of the RTI installation. Only connection requests occur over the server ports, as with any TCP/IP system. For information about calculating connection requirements and configuring the system accordingly, see the GemFire documentation: Making Sure You Have Enough Sockets.

**Server ports**

Server ports allow client applications to establish connectivity to server instances (ingest grid or distribution grid applications). Because locators provide the ability to dynamically
discover server ports, there is no default value. However, if client applications must traverse a firewall, specify these ports with the following RTISH command:

```bash
rti set property --instances <DistOrIngestGridInstanceName>@<Hostname|*> --name rti.cache-server.port --value 20000
```

### TCP port

This port is required to support TCP communications for peer-to-peer data replication and any other GemFire peer-to-peer traffic. The port value defaults to 0. Because 0 is an invalid port, this value is used as a token for the option to use a random (valid) listener port. The membership port range is used: 1024 to 65535.

Dynamically assigned ports work because RTI cluster members are typically behind the firewall on a secured local LAN. However, if necessary, you can set the port value with the following RTISH command:

```bash
rti set property --instances <DistOrIngestGridInstanceName>@<Hostname|*> --name rti.gemfire.tcp-port --value 20001
```

---

### Configuring a single-host system

This section shows how to use RTISH to configure and run a functioning RTI system in a minimal single-host environment. This type of installation is useful for test installations and developer installations.

The example covers the following tasks:

1. Creating locator and component instances. A minimal installation requires the following instances:
   a. Two Locator instances, one per grid
   b. An ingest grid instance
   c. A distribution grid instance
   d. A Provisioning Manager instance
2. Enabling all of the instances.
3. Setting application and environment properties appropriate to each instance.
4. Installing the license.
5. Starting the instances in the required order.
6. Verifying the status of the system.

---

### Creating applications

A minimal RTI system requires the following applications, including two locators, one for each grid. The system also requires an ingest grid, a distribution grid, and a Provisioning Manager. For example:

```bash
rti create application --type locator --name ingest-grid-locator
rti create application --type locator --name dist-grid-locator
```
rti create application --type ingestgrid --name ingestgrid-one
rti create application --type distgrid --name distgrid-one
rti create application --type provisioning --name provisioning-one

Start the RTISH command-line interface to run these commands.

**Enabling the instances**

Enable an instance for each application that you created in step 1. On a single-node system, instances are all created and enabled on that host (@*). In a multi-host system, this step enables instances on specific hosts. Enabling an instance of an application is equivalent to specifying where the application will run, like a service.

rti enable instance --instances ingest-grid-locator@*
rti enable instance --instances dist-grid-locator@*
rti enable instance --instances ingestgrid-one@
rti enable instance --instances distgrid-one@
rti enable instance --instances provisioning-one@

**Configuring the locators**

Setting properties for the instances is a more detailed task because the requirements are different for each component. The following sections cover the properties required for all of the instances you enabled, starting with the locators. For details about specific properties ("keys" and "values"), see Configuration Properties.

Each system component must connect to the GemFire locator(s) that are configured. Use the wildcard value @* to identify the local host address in these examples.

In the following commands, ensure you are using locator ports that are not already in use. If you already created some test applications and instances, be sure that there are no port conflicts and that there are enough local system resources to run them at the same time.

**Configuring the ingest grid locator**

Set the following properties for the instance named ingest-grid-locator:

rti set property --instances ingest-grid-locator@* --name rti.locator.port --value 10334
rti set property --instances ingest-grid-locator@* --name rti.gemfire.remote-locators --value 127.0.0.1[10335]
rti set property --instances ingest-grid-locator@* --name rti.gemfire.distributed-system-id --value 1
rti set property --instances ingest-grid-locator@* --name rti.gemfire.mcast-port --value 0
rti set property --instances ingest-grid-locator@* --name rti.gemfire.bind-address --value 127.0.0.1
rti set property --instances ingest-grid-locator@* --name rti.locator.bind-address --value 127.0.0.1
rti set property --instances ingest-grid-locator@* --name gemfire.jmx-manager-bind-address --value 127.0.0.1
rti set property --instances ingest-grid-locator@* --name gemfire.jmx-manager-hostname-for-clients --value 127.0.0.1
Configuring the distribution grid locator

Set the following properties for the instance named dist-grid-locator:

```bash
rti set property --instances dist-grid-locator@* --name
rti.locator.port --value 10335
rti set property --instances dist-grid-locator@* --name
rti.gemfire.distributed-system-id --value 2
rti set property --instances dist-grid-locator@* --name
rti.gemfire.mcast-port --value 0
rti set property --instances dist-grid-locator@* --name
rti.locator.bind-address --value 127.0.0.1
rti set property --instances dist-grid-locator@* --name
rti.locator.bind-address --value 127.0.0.1
rti set property --instances dist-grid-locator@* --name
gemfire.jmx-manager-http-port --value 9090
rti set property --instances dist-grid-locator@* --name
gemfire.jmx-manager-port --value 2099
rti set property --instances dist-grid-locator@* --name
gemfire.jmx-manager-bind-address --value 127.0.0.1
rti set property --instances dist-grid-locator@* --name
gemfire.jmx-manager-hostname-for-clients --value 127.0.0.1
```

Configuring the ingest grid

Set properties for the ingest grid, including connectivity (locator) settings:

```bash
rti set property --instances ingestgrid-one@* --name
rti.gemfire.locators --value 127.0.0.1[10334]
rti set property --instances ingestgrid-one@* --name
rti.gemfire.remote-locators --value 127.0.0.1[10335]
rti set property --instances ingestgrid-one@* --name
rti.distgrid.pr.buckets --value 37
rti set property --instances ingestgrid-one@* --name
rti.gemfire.distributed-system-id --value 1
rti set property --instances ingestgrid-one@* --name
rti.gemfire.bind-address --value 127.0.0.1
```

The following properties set the four telecommunications protocols (adr, gb, iucs, iups) as enabled and define other general properties.

```bash
rti set property --instances ingestgrid-one@* --name
rti.ingester.active --value true
rti set property --instances ingestgrid-one@* --name adr.enabled --value true
rti set property --instances ingestgrid-one@* --name gb.enabled --value true
rti set property --instances ingestgrid-one@* --name iucs.enabled --value true
rti set property --instances ingestgrid-one@* --name iups.enabled --value true
```
Chapter 3: Configuring an RTI System

The following properties set the ingest grid filters and enrichers. In this example, all event handling is turned off on ingestgrid-one:

```
rti set property --instances ingestgrid-one@* --name rti.ingester.enrichers.tracer --value false
rti set property --instances ingestgrid-one@* --name rti.ingester.enrichers.blacklist --value false
rti set property --instances ingestgrid-one@* --name rti.ingester.filters.imsi --value false
rti set property --instances ingestgrid-one@* --name rti.ingester.filters.eventTime --value false
rti set property --instances ingestgrid-one@* --name rti.ingester.filters.location --value false
rti set property --instances ingestgrid-one@* --name rti.ingester.filters.refData --value false
rti set property --instances ingestgrid-one@* --name rti.ingester.filters.blacklist --value false
rti set property --instances ingestgrid-one@* --name rti.ingester.filters.globalOptIn --value false
rti set property --instances ingestgrid-one@* --name rti.ingester.filters.globalOptIn.useStartTime --value false
```

Configuring the distribution grid

Set properties for the distribution grid, including setting the locator to the second port (10335), and setting AMQP (Advanced Message Queuing Protocol) properties:

```
rti set property --instances distgrid-one@* --name rti.gemfire.locators --value localhost[10335]
rti set property --instances distgrid-one@* --name rti.gemfire.bind-address --value 127.0.0.1
rti set property --instances distgrid-one@* --name rti.cache-server.bind-address --value 127.0.0.1
rti set property --instances distgrid-one@* --name rti.gemfire.distributed-system-id --value 2
rti set property --instances distgrid-one@* --name rti.distgrid.gateway-receiver.bind-address --value 127.0.0.1
rti set property --instances distgrid-one@* --name rti.gateway-receiver.hostname-for-senders --value 127.0.0.1
rti set property --instances distgrid-one@* --name D.rti.gemfireGatewayReceiverHostNameForSenders --value 127.0.0.1
rti set property --instances distgrid-one@* --name rti.distgrid.pr.buckets --value 37
rti set property --instances distgrid-one@* --name rti.distgrid.amqp-batch.num-seCONDS-timEout --value 10
rti set property --instances distgrid-one@* --name rti.distgrid.amqp-batch.num-events --value 10
```

Set the following general properties for the Provisioning Manager:
rti set property --instances provisioning-one@* --name rti.gemfire.locators --value 127.0.0.1[10335]
rti set property --instances provisioning-one@* --name rti.rabbit.addresses --value 127.0.0.1

The Provisioning Manager must have connectivity to the distribution grid and the host for the (separately installed) RabbitMQ Server. We assume here that rabbitmq-server has been previously installed on the local host.

Configure specific environment variables for all instances:

rti set env --instances ingest-grid-locator@* --name RTI_OPTS --value "-Xmx128m -Xms128m -XX:+UseParNewGC -XX:+UseConcMarkSweepGC -XX:CMSInitiatingOccupancyFraction=50"
rti set env --instances dist-grid-locator@* --name RTI_OPTS --value "-Xmx128m -Xms128m -XX:+UseParNewGC -XX:+UseConcMarkSweepGC -XX:CMSInitiatingOccupancyFraction=50"
rti set env --instances ingestgrid-one@* --name RTI_OPTS --value "-Xmx3g -Xms3g -XX:+UseParNewGC -XX:+UseConcMarkSweepGC -XX:CMSInitiatingOccupancyFraction=50"
rti set env --instances distgrid-one@* --name RTI_OPTS --value "-Xmx3g -Xms3g -XX:+UseParNewGC -XX:+UseConcMarkSweepGC -XX:CMSInitiatingOccupancyFraction=50 -Xdebug -Xrunjdwp:server=y,transport=dt_socket,address=4000,suspend=n"
rti set env --instances provisioning-one@* --name RTI_OPTS --value "-Xmx256m -Xms256m -XX:+UseParNewGC -XX:+UseConcMarkSweepGC -XX:CMSInitiatingOccupancyFraction=50"
rti set env --instances ingester-one@* --name RTI_OPTS --value "-Xmx512m -Xms512m -XX:+UseParNewGC -XX:+UseConcMarkSweepGC -XX:CMSInitiatingOccupancyFraction=50"

You must run RTI with a valid license. Install the license file using the RTISH command:

rti license deploy as shown below. To verify that the license is installed, run the rti license audit command or manually checking the rti-t folder for the file. Refer to Chapter 16 RTISH Commands for the command details.

rti license deploy --file path_to_file/license_file_name.lic

In this step you will start the instances in order, locators first, then the two grids, then the Provisioning Manager. You can use a single rti system start command, which enforces this order by default, or you can use a sequence of rti start commands:
rti system start --clientStartupTimeDelay 10

OR

rti start --instances ingest-grid-locator@*,dist-grid-locator@
sleep --delay 10000
rti start --instances ingest-grid-one@*,distgrid-one@
sleep --delay 10000

For node discovery purposes, the locators must be started first. The delays ensure that instances for different applications are fully started before the next start command is run. After starting the instances, check the system status:

rti start --instances ingest-grid-locator@*,dist-grid-locator@
sleep --delay 10000
rti start --instances ingest-grid-one@*,distgrid-one@
sleep --delay 10000
rti start --instances provisioning-one@

If you run into any problems, you can check the RTI log files for errors. See Managing the RTI System.

Configuring a multi-host system

This section contains a detailed example of a multi-node RTI system and the steps to configure it. This example consists of an eight-node cluster in a single data center. The RTI components are distributed as follows:

- A separate master node runs RTISH and deploys the software to other nodes
- A single ingest grid resides on a separate node
- The ingest grid locator resides on a separate node
- A REST server resides on a separate node
- Two distribution grids reside on separate nodes (RabbitMQ is installed on one of these nodes)
- The Provisioning Manager resides on a separate node
- One node is unused

This section contains a detailed example of a distributed RTI system, including two separate server clusters (and their associated client applications) interconnected via a GemFire WAN Replication Gateway. For simplicity, the example is configured to run on hosts residing on a local LAN in a single data center.

However, the two distinct clusters can be deployed in different data centers and across different geographies. The example is deployed to a set of eight networked hosts. The RTI components are distributed as follows:

- A separate master node
- An Ingestion Cluster comprised of:
An ingest grid server application instance on a dedicated host, configured with a WAN Gateway Sender to forward events to the Distribution Cluster

- A GemFire Locator process for cluster discovery services on a dedicated host

- A Distribution Cluster comprised of:
  - Two distribution grid server application instances, each on a separate host. These hosts are identically configured (except for their local bind-address property), demonstrating both horizontal scalability and active/active continuous availability.
  - Two GemFire Locator processes for cluster discovery services. These are deployed to the same hosts as the distribution grid server application instances (a common practice because GemFire Locators are very lightweight processes). This redundant configuration is typical for production environments requiring continuous availability.
  - A Provisioning Manager application instance deployed to a dedicated host.

**General notes**

You can manage remote hosts by using RTISH. The `rti deploy host` command re-creates the RTI tarball on a remote host and extracts it. Use the deploy command for each machine that will be part of your RTI topology. When RTISH is first started, the local host is automatically added to the set of RTI-managed nodes. Then you can add hosts to the system explicitly by deploying to them. Be sure to use the actual IP address or hostname of the host you want to deploy to as well as an existing deployment path on the remote host. For example:

```
rti deploy host --host rti-node9 --dir /opt/emc/
```

RTISH connects to remote hosts using SSH and SCP. To ensure that all hosts are accessible from RTISH, make sure that SSH is installed on both the local and remote machines and that the SSH server is running on the remote host. Also, create a key pair, place the private key on the machine running RTISH, and deploy the public key to all the hosts you want to control from RTISH. See [Installing the EMC RTI Software](#).

Use the following command to list keys on the local machine:

```
rti list ssh-keys
```

You can also set the passphrase for a particular key:

```
rti set ssh-key password --key ...
```

Once RTI is deployed, you can start and stop components on the remote host. You can set environment variables as required for individual instances. For example:

```
rti create application --type locator --name basic-locator-1
rti set env --instances basic-locator-1@192.168.0.100 --name RTI_OPTS --value "-Xmx2g -XX:PermSize=128m"
rti set property --instances basic-locator-1@192.168.0.100 --key rti.locator.port --value 10334
rti set property --instances basic-locator-1@192.168.0.100 --key rti.locator.host --value 127.0.0.1
```
rti enable instance --instances basic-locator-1@192.168.0.100
rti start --instances basic-locator-1@192.168.0.100
rti system shutdown

You can also undeploy RTI from individual hosts. For example:

rti undeploy host --host rti-node9

Creating the applications

This example of a multi-node RTI system requires the following applications:

- Three locators
- An Ingestor Grid
- Two distribution grids
- A Provisioning Manager

The following commands create the required set of applications:

rti create application --type locator --name Locator1IngestGrid --
description IngestGridLocatorInstance1
rti create application --type locator --name Locator1DistGrid --
description DistributionGridLocatorInstance1
rti create application --type locator --name Locator2DistGrid --
description DistributionGridLocatorInstance2
rti create application --type ingestgrid --name IngestGrid1 --
description IngestGridInstance1
rti create application --type distgrid --name DistGrid1 --
description DistributionGridInstance1
rti create application --type distgrid --name DistGrid2 --
description DistributionGridInstance2
rti create application --type provisioning --name ProvisioningApp
--description Provisioning-one

Setting general ingestor properties

Set the following properties for the ingest grid application. These properties enable or
disable event protocols and filters.

rti set property --instances IngestGrid1@* --name
rti.ingester.active --value true
rti set property --instances IngestGrid1@* --name adr.enabled --
value true
rti set property --instances IngestGrid1@* --name gb.enabled --
value true
rti set property --instances IngestGrid1@* --name iucs.enabled --
value true
rti set property --instances IngestGrid1@* --name iups.enabled --
value true
rti set property --instances IngestGrid1@* --name
rti.ingester.enrichers.tracer --value false
rti set property --instances IngestGrid1@* --name
rti.ingester.enrichers.blacklist --value false
rti set property --instances IngestGrid1@* --name rti.ingester.filters.imsi --value false
rti set property --instances IngestGrid1@* --name rti.ingester.filters.eventTime --value false
rti set property --instances IngestGrid1@* --name rti.ingester.filters.location --value false
rti set property --instances IngestGrid1@* --name rti.ingester.filters.refData --value false
rti set property --instances IngestGrid1@* --name rti.ingester.filters.blacklist --value false
rti set property --instances IngestGrid1@* --name rti.ingester.filters.globalOptIn --value false
rti set property --instances IngestGrid1@* --name rti.ingester.filters.globalOptIn.useStartTime --value false

Set the following properties for the ingest grid and distribution grid:

rti set property --instances IngestGrid1@* --name rti.ingestgrid.pr.buckets --value 37
rti set property --instances IngestGrid1@* --name rti.distgrid.pr.buckets --value 37
rti set property --instances DistGrid1@* --name rti.distgrid.pr.buckets --value 37
rti set property --instances DistGrid2@* --name rti.distgrid.pr.buckets --value 37

GemFire breaks data into units of storage known as buckets. The rule of thumb for the number of buckets is at least 10 times the number of servers in the cluster. This allows the GemFire Distributed Resource Manager enough flexibility to rebalance data as needed, which occurs at the granularity level of one bucket at a time. The default buckets value is 37.

Note: The property rti.distgrid.pr.buckets must be set on BOTH the ingest grid and the distribution grid. On the ingest grid, it is used by the single-hop event router to determine each event's correct remote bucket Id (and associated single-hop route). This does not imply that rti.distgrid.pr.buckets and rti.ingestgrid.pr.buckets are required to be the same as each may have a different and appropriate value depending on the respective cluster sizes (in other words, the number of deployed/enabled instances of each grid application type).

The target distributed environment has multiple network interface cards (NICs) for each host. In a performance optimized environment, we might choose different bind-address values to segregate different traffic flows (for example, client/server, peer-to-peer, or WAN Gateway Replication). Alternatively, multiple physical NICs might be bonded to create a single, high-bandwidth, virtual network interface. Another consideration is the visibility of addresses and hostnames (using fully qualified domain names) across WAN connections, which are typically different from what is configured on the local LAN.

For simplicity, and because all components are deployed to a single LAN in this example, a single set of NICs is used to map to the same local LAN. These all begin with the following three octets: 10.110.124. For deployments to different data centers, the
components in this example are split up and managed by separate (and local) instances of RTISH, with the only direct interaction occurring at runtime through the WAN Gateway connectivity and runtime data replication.

The hostname/IP address mapping for the hosts in this example is as follows:

ogg-w9: 10.110.124.105
ogg-w10: 10.110.124.106
ogg-w11: 10.110.124.107
ogg-w12: 10.110.124.108
ogg-w13: 10.110.124.109
ogg-w14: 10.110.124.110
ogg-w16: 10.110.124.112

Set the following properties to establish connectivity among the RTI components:

# Shared Connectivity Properties for Ingester, Ingest Grid & Ingest Grid Locator
rti set property --instances IngestGrid1@*,Locator1IngestGrid@*
   --name rti.gemfire.locators --value 10.110.124.107[11334]

# Shared Properties for Ingest Grid and Ingest Grid Locator
rti set property --instances IngestGrid1@*,Locator1IngestGrid@*
   --name rti.gemfire.remote-locators
   --value 10.110.124.109[10334],10.110.124.108[10334]
rti set property --instances IngestGrid1@*,Locator1IngestGrid@*
   --name rti.gemfire.distributed-system-id
   --value 1

# Shared Properties for Distribution Grid, Distribution Grid Locators, and Provisioning Manager
rti set property --instances
   DistGrid1@*,DistGrid2@*,Locator1DistGrid@*,Locator2DistGrid@*,ProvisioningApp@
   --name rti.gemfire.locators
   --value 10.110.124.109[10334],10.110.124.108[10334]

# Shared Properties for Distribution Grid and Distribution Grid Locators
rti set property --instances
   DistGrid1@*,DistGrid2@*,Locator1DistGrid@*,Locator2DistGrid@
   --name rti.gemfire.distributed-system-id
   --value 2

# Shared Properties for Distribution Grid instances only
rti set property --instances DistGrid1@*,DistGrid2@
   --name rti.rabbit.addresses --value 10.110.124.108

# Distribution Grid Locator 1
rti set property --instances Locator1DistGrid@
   --name rti.locator.host --value 10.110.124.109
rti set property --instances Locator1DistGrid@*
  --name rti.locator.port --value 10334
rti set property --instances Locator1DistGrid@*
  --name rti.gemfire.bind-address --value 10.110.124.109

# Distribution Grid Locator 2
rti set property --instances Locator2DistGrid@*
  --name rti.locator.host --value 10.110.124.108
rti set property --instances Locator2DistGrid@*
  --name rti.locator.port --value 10334
rti set property --instances Locator2DistGrid@*
  --name rti.gemfire.bind-address --value 10.110.124.108

# Distribution Grid Instance 1
rti set property --instances DistGrid1@*
  --name rti.gateway-receiver.bind-address --value 10.110.124.108
rti set property --instances DistGrid1@*
  --name rti.gemfire.bind-address --value 10.110.124.108
rti set property --instances DistGrid1@*
  --name rti.cache-server.bind-address --value 10.110.124.108
rti set property --instances DistGrid1@*
  --name rti.gateway-receiver.hostname-for-senders
    --value 10.110.124.108
rti set property --instances DistGrid1@*
  --name D.rti.gemfire.GatewayReceiver.HostNameForSenders
    --value 10.110.124.108
rti set env --instances DistGrid1@* --name RTI_OPTS
  --value "-XX:PermSize=128m -Xmx12g -Xms12g"

# Distribution Grid Instance 2
rti set property --instances DistGrid2@*
  --name rti.gateway-receiver.bind-address --value 10.110.124.109
rti set property --instances DistGrid2@*
  --name rti.gemfire.bind-address --value 10.110.124.109
rti set property --instances DistGrid2@*
  --name rti.cache-server.bind-address --value 10.110.124.109
rti set property --instances DistGrid2@*
  --name rti.gateway-receiver.hostname-for-senders
    --value 10.110.124.109
rti set property --instances DistGrid2@*
  --name D.rti.gemfire.GatewayReceiver.HostNameForSenders
    --value 10.110.124.109
rti set env --instances DistGrid2@* --name RTI_OPTS
  --value "-XX:PermSize=128m -Xmx12g -Xms12g"

# Ingest Grid Instance
rti set property --instances IngestGrid1@*
  --name rti.gemfire.bind-address --value 10.110.124.106
rti set property --instances IngestGrid1@*
Deploying the configuration to the hosts

Deploy the RTI configuration to all of the other hosts in the system:

```bash
rti deploy host --host ogg-w9 --dir /opt/emc/
rti deploy host --host ogg-w10--dir /opt/emc/
rti deploy host --host ogg-w11--dir /opt/emc/
rti deploy host --host ogg-w12--dir /opt/emc/
rti deploy host --host ogg-w13--dir /opt/emc/
rti deploy host --host ogg-w14--dir /opt/emc/
rti deploy host --host ogg-w15--dir /opt/emc/
```

Enabling the applications on the hosts

Enable an instance of each configured application on the appropriate host machine:

```bash
rti enable instance --instances IngestGrid1@ogg-w10
rti enable instance --instances Locator1IngestGrid@ogg-w11
rti enable instance --instances DistGrid1@ogg-w12
rti enable instance --instances Locator2DistGrid@ogg-w12
rti enable instance --instances Locator1DistGrid@ogg-w13
rti enable instance --instances DistGrid2@ogg-w13
rti enable instance --instances ProvisioningApp@ogg-w14
```

Installing the license

You must run RTI with a valid license. Install the license file on all the hosts using the RTISH command `rti license deploy` as shown below. This command should be run on the master node. The file gets installed on all RTI hosts including the master node. You can verify the license is installed by running the `rti license audit` command or manually checking the `rti-t` folder for the file. Refer to the chapter RTISH Commands for the command details.

```bash
rti license deploy --file path_to_file/license_file_name.lic
```

Starting the system

Start all instances of the multi-host RTI system. You can use a single `rti system start` command, which starts all applications in order, or you can use a sequence of `rti start` commands:

```bash
rti system start
```

```
OR
```
rti start --instances Locator1DistGrid@ogg-w13, Locator2DistGrid@ogg-w12
sleep --delay 3000
rti start --instances Locator1IngestGrid@ogg-w11
sleep --delay 3000
rti start --instances DistGrid1@ogg-w12, DistGrid2@ogg-w13
sleep --delay 3000
rti start --instances IngestGrid1@ogg-w10
sleep --delay 10000
rti start --instances ProvisioningApp@ogg-w14

Now check system status:

rti status --instances IngestGrid1@ogg-w10
rti status --instances Locator1IngestGrid@ogg-w11
rti status --instances Locator2DistGrid@ogg-w12
rti status --instances DistGrid1@ogg-w12
rti status --instances DistGrid2@ogg-w13
rti status --instances Locator1DistGrid@ogg-w13
rti status --instances ProvisioningApp@ogg-w14
Configuring output streams

By default, the RTI system sends output from the Distribution Grid to two different streams:

- Location
- Transaction

As an alternative to these standard output streams, you can configure output streams that contain any combination of the following fields:

- imsi
- msisdn
- imei
- mccmnc
- transactionTarget
- firstLac
- firstCellTower
- lac
- cellTower
- timeUTC
- startTimeUTC
- eventDetail
- eventType
- eventStatus
- protocolName
- latitude
- longitude
- radii90
- device
- ingestTime
- processedTime
- correlationBatchId
- correlationIndex
- temporaryId

For descriptions of these fields, Table 93 in Output format for event streams.

While most of the above fields are values passed through the system from the original input events, correlationBatchId and correlationIndex are metadata fields added by RTI. When RTI receives events from upstream system, they are sometimes received as a
batch. RTI automatically splits these batches into individual events for downstream processing and consumption. For applications that consume these events to be able to reconstruct the original batches, we have made these two new fields available.

`correlationBatchId` is a unique UUID generated by the RTI system. Events that originate from the same upstream batch of events share a unique `correlationBatchId`. `correlationIndex` provides the ordered position of the event within the original batch received from an upstream event publisher. `correlationIndex` is a zero-based index.

The field `temporaryId` carries the values for the temporary identifier. The Spring Expression Language (SpEL) expression to use in the output format is `temporaryId`.

**Note:** Depending on the protocol, the `temporaryId` may contain a different parameter such as `tmsi`, `p-tmsi`, or `guti`, and so on. Refer to the data dictionary for the parameter from the protocol specification that maps to `temporaryId`.

You can configure custom output streams in two different ways:

- Set the `rti.distgrid.format.csvField.format` property in the distribution grid.
- Use the `rti output format` command in RTISH.

**Setting the `rti.distgrid.format.csvField.format` property**

You can use this property to define a custom set of fields under the output stream name `csvField`. You set the `rti.distgrid.format.csvField.format` property in the distribution grid. Specify a list of comma-separated field names as follows:

```
rti.distgrid.format.csvField.format=imsi,msisdn,imei,lac
```

**Using the `rti output format` command**

You can use the `rti refdata outputformat upsert --file` command in RTISH to load a JSON file that supplies the format definition, in terms of a name and a list of fields for each output stream. You can use one file to load definitions for multiple output streams. For example:

```json
[

{
  "outputFormatID" : "newformat", "outputFormatConfiguration" : "imsi,imei,lac,cellTower"
},

{ "outputFormatID" : "special", "outputFormatConfiguration" : "imsi,imei,lac,eventDetail.CallType,cellTower"
}
]
```

The second output format in this example uses an example of the `eventDetail.name` structure to refer to a protocol-specific subfield in the Transaction Status field; see Output format for event streams for information about the available fields.

If you are setting up an Event Stream Subscription (ESS) using the Provisioning Manager, ensure that the ESS message contains the correct output format name. For example, in
the following message payload, the SUB05 subscription is configured to receive the output format named newformat:

SUB05,false,2012-11-23T00:00:00.000+0000,2099-12-01T23:59:59.999+0000,newformat

**Setting formats for the transaction and location output files**

You can use the following properties to override the default behavior of the transaction and location streams.

```plaintext
rti.distgrid.lands.format=location
rti.distgrid.lors.format=transaction
```

These properties accept new format names that you have defined with the `rti output format` command. For example:

```plaintext
rti.distgrid.lors.format=newformat
```

In this case, the standard transaction output files produce the set of fields that you defined as `newformat`. You can also set these properties to `csvField`. The resulting output file directory on the distribution grid is named accordingly (taking the name of your new format or `csvField`).

### Filtering output streams

**Note:** Beginning with the RTI 3.0 release, refer to the *Reference* chapter in the *EMC Real-Time Intelligence (RTI) Streaming Guide* to deploy filtered streams using the new approach.

**JSON/SpEL based filter expressions for LORS output streams**

You can define and deploy output filters for RTI Location OR Subscriber (LORS) streams, receiving from RTI only a subset of events that pass the basic LORS filter. As with standard LORS output, these filtered streams can be configured to output either to a file or to AMQP. RTISH has also been enhanced to manage the deployment and enabling of filtered streams.

**Filtered stream definition format**

Stream definitions are defined using a JSON payload that conforms to a schema. This section provides an example and description of the schema attributes.

Here is a sample filtered stream definition:

```json
{  "name" : "example1",  "enabled" : true,  "where" : "latitude > 0",  "outputType" : "amqp",  "amqpExchange" : "DECODER.STREAM",  "amqpRoutingKey" : "DECODER.STREAM.FILTERED.EXAMPLE1" }
```

This stream will output all events with latitude greater than 0 to AMQP.
### Filtered stream attributes

**Table 3. Filtered stream attributes**

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Description</th>
<th>Datatype, etc</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>All streams require a unique name that is be used throughout the system. This name is used to execute stream management functions such as deploy/undeploy, enable/disable, and describe.</td>
<td>String</td>
</tr>
<tr>
<td>enabled</td>
<td>Flag to determine if the stream processes events or not.</td>
<td>Boolean (true/false)</td>
</tr>
<tr>
<td>where</td>
<td>A SpEL expression that defines the filter conditions. If the expression resolves to true, the message proceeds down the stream. An unspecified &quot;where&quot; attribute allows all messages down the stream. Expressions are based upon the SpEL syntax.</td>
<td>SpEL construct</td>
</tr>
<tr>
<td>outputType</td>
<td>A required attribute that indicates the stream's target output type. Currently AMQP and file are supported.</td>
<td>String</td>
</tr>
<tr>
<td>amqpExchange</td>
<td>When outputType is set to amqp, this attribute determines which AMQP exchange to stream will output to.</td>
<td>String</td>
</tr>
<tr>
<td>amqpRoutingKey</td>
<td>When outputType is set to amqp, this determines the routing key used by AMQP.</td>
<td>String</td>
</tr>
<tr>
<td>amqpBatchSize</td>
<td>Batching will send multiple events in a single AMQP message to improve overall performance. Adjust based on performance needs. A value of 1 essentially disables batching.</td>
<td>Integer greater than 0</td>
</tr>
<tr>
<td>amqpBatchTimeout</td>
<td>The current batch of messages are sent either on this timeout or when reaching the batch size, whatever comes first.</td>
<td>Integer no less than 100. The timeout is measured in milliseconds. A value of at least 500 is recommended.</td>
</tr>
</tbody>
</table>
### Configuring an RTI System

#### Attribute Name | Description | Datatype, etc
---|---|---
fileBasePath | When outputType is set to file, this determines the location to which the files will be saved. Setting this value to "$(rti.data)/example1" saves the output files to rti-t/var/data/[dist-grid-name]/example1. | String. Must resolve to a valid directory in the locally mounted file system. It is very highly recommended that the file system be a local disk drive as RTI can overwhelm shared disk infrastructure. If any SAN technology is used, consult with RTI support and be sure to soak test the system at full load.

fileDateFormat | Each output file name is appended with a timestamp. This property allows users to customize this timestamp. | Example: yyyyMMdd'T'HHmmssSSS

fileBatchSize | Batching writes multiple events to the file at once to improve overall performance. Adjust based on performance needs. A value of 1 essentially disables batching. | Integer greater than 0

fileBatchTimeout | The current batch of messages is flushed/written to file either on this timeout or when reaching the batch size, whatever comes first. | Integer no less than 100. The timeout is measured in milliseconds. A value of at least 500 is recommended.

select | This field can take on several constructs: SpEL, Cascading Hash Expression, Output Transformer class and configuration | Data type can be SpEL, cascading hash, or output transformer class.

selectType | Indicates how to interpret the text that is contained in the select statement. | Valid values are SPEL, CLASS, and CASCADE_HASH. Default is CASCADE_HASH.

---

**Note**: Attributes beginning with "amqp" are only applicable when outputType equals "AMQP" and attributes beginning with "file" are only applicable when outputType equals "file".

### Using RTISH to manage filtered streams

A handful of management tasks for Filtered Streams are provided by RTISH:

- `rti stream deploy --definition <path to JSON stream definition file>`
This command deploys a named stream. The stream name is determined by the "name" attribute value assignment contained within the definition file, and is needed for subsequent management tasks. For our RTISH command descriptions, we assume use of the filter definition named "example1" provided in the sample above.

- **rti stream undeploy --name example1**
  Undeploys a previously deployed filtered output stream.

- **rti stream disable --name example1**
  Disables a previously deployed filtered output stream so that it stops outputting events.

- **rti stream enable --name example1**
  Enables a previously disabled filtered output stream (it will resume outputting events).

- **rti stream describe --name example1**
  Prints out a description of the stream and all its attributes.

- **rti stream list**
  Lists all deployed filtered streams on the system, as well as their enabled/disabled status.

**Maximum Java virtual machine size**

EMC recommends a maximum Java virtual machine (JVM) process size of 120 GB or less. This limit is due to the libc implementation used with Java. Calculate the maximum Java heap using the following calculation:

\[
\text{Total JVM process memory} = \text{-Xmx} - \text{XX:MaxPermSize} + (\text{<NumberOfConcurrentThreads>*-Xss}) + \text{<OtherMemory>}.\]

Other memory must be a constant of about 2G. This memory is required for Java to manage the JVM heap. EMC recommends that you leave ample headroom. The recommended maximum size of the memory allocation pool is 96 GB (-Xmx 96GB).

Reserve some memory for the operating system so that it can operate without swapping data with the hard drive.

**Note:** Do not configure larger heap sizes because additional or unused memory is available on a host. Configuring too high a heap size can have consequences, such as delaying CMS GC cycles for so long that potentially constrained system resources are not cleaned up for a long time, and might eventually be exhausted. For example, file descriptors are created and destroyed over time by the normal operation of the system, but whose (already high by default) limit might be breached after several days of no clean up. Unnecessarily large heap sizes might also cause long GC pauses that do not occur if the heap size is smaller or if the heap is used and collected regularly.
Chapter 3: Configuring an RTI System

Using cascading hash

Cascading hash can be used to create high performance user-defined output. The cascading hash first looks for the field in the protocolDetailMap. If the entry is not found, it then looks in the protocolAttributeMap. If the field is not found, cascading hash uses reflection to attempt to find the entry.

To configure cascading hash, refer to the select and selectType attributes in Filtered Stream Attributes.

Cascading hash example

The following example shows how to write International Mobile Subscriber Identity (IMSI), latitude, and longitude using the cascading hash method:

```json
{
    "name" : "user_defined_with_select",
    "enabled" : true,
    "where" : "latitude > 0",
    "select" : "imsi,latitude,longitude",
    "outputType" : "amqp",
    "amqpExchange" : "DECODER.STREAM",
    "amqpRoutingKey" : "DECODER.STREAM.SELECT.USER_DEFINED",
    "amqpBatchSize" : 1,
    "amqpBatchTimeout" : 100
}
```
This chapter presents the following topics:

- **Loading reference data** ................................................................. 46
- **RDL error messages** ..................................................................... 61
Loading reference data

The operating company decides what reference data they need to support their consumers. You can disable all of the ingest grid filters for testing purposes, in which case you do not need to load any reference data, except organization units, but this is an exceptional case. Without reference data, you can check that events flow through the system, but you cannot filter or enrich the events. For example, if you enable the filter for cell towers, you must load cell tower data. Most operating companies will need to load most or all of the data types in Table 4. Subscriber profiles may not be necessary if you choose to turn off the enricher that maps IMSIs to MSISDNs when they are not available in the source event data.

Event subscriptions, geofences, and linkgeofences are loaded into the system in message format using the Provisioning Manager, not the RDL. The same applies to External opt-ins (location and subscriber).

The RDL is a Java application that functions as a client to the RTI grid servers. RDL provides a command-line shell for loading all types of reference data and running other maintenance operations. You can also use an RTISH command, rti orgunit upsert, to load reference data for organization units.

This section describes the basics of loading reference data. For additional and more detailed information as well as the different methods you can use, refer to:

- Reference data formats
- Chapter 18 Loading Reference Data with RTISH
- Chapter 19 Provisioning Manager Message Interface for Loading Reference Data

Use the RDL to load reference data that is used to qualify events for processing as they stream through the system. You can load large CSV files into GemFire regions for reference during transformation and enrichment of event data. As events stream into the Ingester and Distribution grids, lookups match the events with criteria defined by filters. In turn, events are included in or excluded from further processing and distribution to consumers.

The RTI system relies on the following types of core reference data:

- **Enrichment data**: baseline network information about cell towers, mobile devices, and subscribers. This information enriches raw ingested events by providing specific details about locations, equipment, and users.
- **Filtering data**: opt-ins for locations and subscribers. This information identifies consent to track certain users and locations, which in turn determines whether certain events qualify for distribution.
- **Subscriptions**: data in the form of event stream subscriptions, as requested by predefined organization units (OUs). This information defines the criteria for delivering event streams to third parties, including geofences and additional opt-in filters.
Baseline reference data derives from data sources at the "operating company," that is, the company that manages the RTI system and accepts event subscriptions from external applications. This data is specific to the cellular network, its users, and their mobile devices; therefore it is "anonymized" as part of the routine RTI processing.

You only need to load reference data that is relevant to the requirements of the subscriptions you plan to service. Reference data is loaded from CSV files, modeled in RTI, and stored efficiently in data containers (GemFire regions) on the RTI grids.

The RTI system also supports three different feeds or streams: Location AND Subscriber (LANDS), LORS, and data record (DR). See the discussion of opt-ins and opt-in types.

In the loader commands, you have to specify the reference data type (--datatype), and for opt-in data types, the opt-in type as well (--optintype). The following data types are supported:

---

**Reference data types**

---
### Chapter 4: Loading Reference Data

**Table 4. Reference data types**

<table>
<thead>
<tr>
<th>Datatype</th>
<th>Purpose</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CellTower</td>
<td>Enrichment</td>
<td>Network topology data. One cell tower is represented per data record, with its ID, name, and information about its location in terms of geospatial coordinates.</td>
</tr>
<tr>
<td>Device</td>
<td>Enrichment</td>
<td>Mobile device information: model, manufacturer, type allocation code (TAC), device generation (2G, 3G, or 4G), UA profile.</td>
</tr>
<tr>
<td>Geofence</td>
<td>Subscription</td>
<td>Geographic boundaries, which are associated with different event subscriptions (a way of filtering events by location from the point of view of third-party applications). Typically loaded in message format as part of an event subscription.</td>
</tr>
<tr>
<td>Linkgeofence</td>
<td>Subscription</td>
<td>An association (link) between a geofence and one or more subscriptions. Typically loaded in message format as part of an event subscription.</td>
</tr>
<tr>
<td>LocationOptin</td>
<td>Filtering</td>
<td>Opt-in that qualifies events by network location (LACs, TACs), cell ID, and time intervals. Location activity can be opted into event streams, as can subscriber activity. See the following discussion of opt-in types.</td>
</tr>
<tr>
<td>Orgunit</td>
<td>Subscription</td>
<td>An organization unit (OU) is an entity that is entitled to access RTI using AMQP. This data type is defined in terms of unique organization and unit names and effective dates.</td>
</tr>
<tr>
<td>SubscriberOptin</td>
<td>Filtering</td>
<td>Opt-in that qualifies events by subscriber IMSI, location area codes (LACs), cell IDs, and time intervals (that is, when and where the subscriber can be tracked). A subscriber’s consent to be localized in the network for a stream is known as a subscriber opt-in. For more information, refer to Opt-ins and opt-in types.</td>
</tr>
<tr>
<td>SubscriberProfile</td>
<td>Enrichment</td>
<td>Subscriber (IMSI) to MSISDN mapping. RTI is also capable of maintaining a mapping between IMSI values and their paired TMSI values. See the Ingester and ingest grid sections in Configuration Properties. A subscriber profile includes one or more subscriber groups that the subscriber belongs to. This data type also serves as an identifier for the subscriber in an external system.</td>
</tr>
<tr>
<td>Subscriptions</td>
<td>Subscription</td>
<td>An event subscription corresponds to one event stream that is delivered to an organization unit. The subscription also defines its own effective dates.</td>
</tr>
<tr>
<td>Trace</td>
<td>Tagging</td>
<td>Trace data is tagged for tracing, which simply means that all events originating from a given location, device, or subscriber are marked.</td>
</tr>
</tbody>
</table>

**Opt-ins and opt-in types**

RDL supports subscriber opt-in and LANDS opt-in data types. An opt-in expresses an interest in events for a certain set of subscribers or locations. You can think of opt-ins as filtering rules that respond to requests for privacy (location-based rules and subscriber-
based rules). Opt-ins may be further filtered with time intervals: not only does a subscriber consent to be tracked, but consents to be tracked during specific times of the day.

**Subscriber opt-ins**

When subscribers to a phone service set up a contract, they indicate their consent for the phone company to track their movements. If they agree, they are "opting in." A corporate contract may opt in on its users’ behalf. When a subscriber opts in, this consent is similar to the consent given to advertisers to send solicitation emails or use personal information on social networks. Subscribers choose to opt in or opt out.

**LANDS opt-ins**

A LANDS opt-in has no knowledge of a subscriber. This rule represents a consent expressed by the network operator, not an individual. The company may opt to target specific locations but may be required to opt out of others. For example, the company may have to comply with government regulations that restrict the tracking of events in certain areas. The company may also want to focus on certain geographic locations for certain campaigns.

**Opt-in types**

Opt-in types are attributes that you can attach to. The following opt-in types are supported:

LocationOptin and SubscriberOptin filters:

- Blacklist
- External
- Global
- LANDS
- LORS

Opt-in types function as a hierarchy, providing up to three levels of visibility or consent. This hierarchy applies to both subscriber and LANDS opt-ins. The global and blacklist types are at the top of the hierarchy. The "feed-specific" or "stream-specific" types are at the next level: LANDS, LORS. The external type is at the lowest level, and is related to the LANDS and LORS types.

![Hierarchy of Opt-in Types](image-url)
**Global opt-in type**

The global opt-in applies early in RTI processing, in the ingest grid. A subscriber, location, or a combination of the two opts in or out "globally" and based on those preferences, the next level of preferences comes into play. This filter may be used from a capacity planning or optimization point of view. An event subscription may be interested in events for a specific city only, so the ingest grid discards all events at the source that do not occur in that location, rather than waste capacity or processing.

**LANDS and LORS opt-in types**

The distribution grid separates events into two streams: **LANDS events** and **LORS events**. If subscribers and locations are opted in globally, they can opt into or out of these two feeds (output types). The location output format contains attributes oriented toward location-based use cases. The transaction format applies to support and quality of service use cases (such as subscriber experience with calls).

- To qualify for further processing on the LANDS stream, events must be filtered with a `LocationOptin` of type `Location` and a `SubscriberOptin` of type `Location`
- To qualify for further processing on the LORS stream, events may be filtered with a `LocationOptin` of type `Transaction` or a `SubscriberOptin` of type `Transaction`

Note that the LANDS event stream is more restrictive, in that it requires both the location and the subscriber to be opted in. LORS events only require either the location or the subscriber to be opted in. The network operator controls the opt-ins for LANDS and LORS events. Subscribers and locations are opted in or out for these streams, and these opt-ins apply to all third-party subscriptions.

**External opt-in type**

External applications (third parties) can set up subscriptions and opt in subscribers and locations (locations are defined using geofences in this case). External opt-ins are set up using Provisioning Manager messages. The network operator may intervene after the fact, by deleting subscribers or geofences using the RDL, but the operator has no control over keeping them filtered out. The network operator does not set up any filtering on the behalf of individual applications that submit event subscriptions.

---

**Important**: External opt-ins should not be loaded into the system unless this approach is explicitly recommended by EMC.

**Blacklist opt-in type**

As a peer of the global type, the blacklist type defines whether a subscriber or location is eligible for inclusion in any event processing. RTI interprets the blacklist attribute as a negation of consent so the subscribers and locations in question are never tracked. A blacklisted record is dropped immediately and cannot be processed regardless of other opt-ins or filters that may be set. Other levels of the hierarchy do not come into play for blacklisted events.

For an example of the RDL commands that you would run to set up opt-ins, see **Loading reference data**. For details about the required formats for supported reference data, see **Reference Data Formats**.
Event output files

Events that flow through the system may be stored in CSV output files on the distribution grid and made available to event stream subscriptions via the AMQP interface (RabbitMQ). See Output format for event streams for details about the contents of the output.

Global subscriber opt-ins control whether any downstream output occurs. Events that qualify for distribution under that opt-in are, at a minimum, written to a transaction output file. Events that also qualify for distribution via the Global location opt-in are also written to a location output file. The output files are written to the following directories:

- /var/data/<DistGrid_instancename>/location
- /var/data/<DistGrid_instancename>/transaction

External opt-ins (LANDS and LORS) determine whether the events that reach the output files may qualify for event stream subscriptions. (This AMQP stream is further filtered by the presence of linkgeofences; see Subscribing to events.)

Trace data

RTI supports three types of trace reference data:

- Device: International Mobile Station Equipment Identity (IMEI) with optional filters on Cell IDs and LACs
- Location: Location Area Code (LAC) with optional filter on Cell IDs
- Subscriber: IMSI with optional filters on "transaction pattern" (bparty), LACs and Cell IDs

Based on the trace data that is loaded, all events originating from a given location (LAC), subscriber, or device are tagged for tracing. Event messages are tagged throughout the system (in the Ingester, ingest grid, distribution grid, and RabbitMQ applications). The event messages are tagged as they flow through the channels and logged for each RTI application. This feature is helpful for discovering where events are discarded as they are processed through the system.

It is also possible to tag an event to be intercepted. Tagged events are logged to a separate intercept log file. While the trace log file will not contain sensitive data (such as the IMSI), the intercept log file will.

Enabling tracing

Enable tracing in the logback.xml configuration file for each application.

The following is an example of the logback.xml file loggers for the ingest grid that includes a logger each configured for trace and intercept.

```xml
<logger name="rti.trace.log" level="INFO">
    <appender-ref ref="TRACE" />
</logger>

<logger name="rti.intercept.log" level="INFO">
    <appender-ref ref="INTERCEPT" />
</logger>
```
Configuring tracing in the ingester

An event that passes through the Ingester is traced in the Spring Integration channels; specifically in the Input and Output channels. Determine which events to trace, if any, using a SpEL expression.

Configure the `rti.ingester.trace.spel` property with the required SpEL expression. The default value is empty. No tracing occurs.

The SpEL can be changed dynamically by using an MBean:

```java
io.pivotal.rti.ingester.json.processor.TraceLoggingConditionalInterceptor
```

This MBean exposes methods to specify the SpEL and enable and disable the interceptor.

![Figure 3. JMX tracing example](image)

Note: The expression includes hashed IMSI values.

To trace events with specific IMSIs, specify a SpEL expression in the following format:

```text
imsi=='imsiValue' || imsi=='imsiValue'
```

For example:

```text
imsi=='99dccd248953e9ec6c9b45646ed34999' ||
imsi=='88dccd248953e9ec6c9b45646ed34888' ||
imsi=='77dccd248953e9ec6c9b45646ed34777'
```

Use the same format to trace specific LACs or TACs:

```text
lac=158 || lac==159 || lac==160
```

You can also trace events with specific cell IDs:

```text
cellId==24501 || cellId==24502
```

To trace IMEI values, specify the obfuscated value to match:

```text
imei=='354262040017XXXX0' || imei=='999262040017XXXX0' ||
imei=='888262040017XXXX0'
```

You can combine SpEL expressions to construct the full expression for matching:
Enabling tracing and intercepting in the ingest grid

Enable tracing and intercepting in the ingest grid by configuring the following properties:

- rti.trace.disable
  Default value: false (enabled)
- rti.intercept.disable
  Default value: true (disabled)

Alternatively, enable tracing and intercepting for subscriber, device and location separately by configuring the following properties:

- Tracing:
  - rti.trace.device.disable
  - rti.trace.location.disable
  - rti.trace.subscriber.disable
- Intercepting:
  - rti.intercept.device.disable
  - rti.intercept.location.disable
  - rti.intercept.subscriber.disable

Tracing in the distribution grid

When tracing is enabled in the ingest grid, applicable events continue to be traced in the distribution grid. A trace log records each Spring Integration message channel that the event passes through.

The trace log file is automatically written to the logs directory in the distribution grid instance. The filename is rti-distgrid.trace.log.

Configure logging in the distribution grid logback.xml file.

The loader can be used to load data into both grids. Before running any load operations, make sure the loader connects to the correct grid by setting rti.gemfire.locators in the common/rdl/etc/config.properties file. This setting must point to the host and port for the locators in the grid. For example:

```
$ more config.properties
...
####################################################
# Gemfire Client
####################################################
## Hosts and ports for Gemfire locators
#rti.gemfire.locators=10.1.2.3[10334]
## Log level for gemfire
#rti.gemfire.log-level=config
...
```
To start the loader, run the following command from the root `rti-t` installation directory:

```bash
$ ./common/rdl/bin/loader
rti_user1 starting 'refdataloader' on xxxxxxx.xxx.xxx.com...
```

Welcome to RTI Reference Data Loader

```
rti>
```

If the following message is displayed, check the configuration and network connectivity:

```
Unable to connect to remote cluster. Check log files for further information.
```

### Types of load

The RDL supports the following load operations:

#### Table 5. RDL load operations

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>create</td>
<td>Delete any current reference data of the specified type, then load new data from the specified file.</td>
</tr>
<tr>
<td>update</td>
<td>Update only the reference data in the specified file (based on the keys in the file). Existing reference data that does not match the keys in the file is left intact.</td>
</tr>
<tr>
<td>delete</td>
<td>Delete only the data that matches the keys in the file for the specified data type. If no file is specified, delete all of the reference data for the specified type.</td>
</tr>
</tbody>
</table>

For example, the following commands load new cell tower data:

```
create --file refdata/cell_tower.csv --datatype CellTower
```

The following examples show commands that update reference data:

```
update --file refdata/location_optin.csv --datatype LocationOptin --optintype Global
update --file refdata/subscriber_optin.csv --datatype SubscriberOptin --optintype Location
update --file refdata/imsi_to_msisdn.csv --datatype SubscriberProfile
```

The following examples delete cell tower reference data:

```
delete --datatype CellTower --file refdata/delete_cell_tower.csv
```

Data of different types is loaded into different regions (`/REF_CellTower` and `/REF_MobileEquipment`, for example).

The RDL supports several other operational commands; see [RDL commands](#).
Loading cell towers

1. Run the following RDL command to load new cell tower data:
   
   \texttt{rdl} > \texttt{create} --file <filename>.csv --datatype CellTower

2. Run the following RDL command to build the spatial index in the distribution grid to make cell towers available for the geofence mechanism:
   
   \texttt{rdl} > \texttt{buildspatialIndex}

The system is now ready to receive geofences via the Provisioning Manager AMQP interface.

Updating cell towers

1. Run the following RDL command to update the reference data for one or more cell towers when the system is running and geofences have been provisioned by third-party users:
   
   \texttt{rdl} > \texttt{update} --file <filename>.csv --datatype CellTower

   Alternatively, to update all cell towers:
   
   \texttt{rdl} > \texttt{create} --file <filename>.csv --datatype CellTower

   The cell towers will now be used by the cell tower enricher, but the spatial indexes and event stream subscriptions are not yet affected.

2. Run the following RDL command to build the spatial index in the distribution grid to make cell towers available for the geofence mechanism.
   
   \texttt{rdl} > \texttt{buildspatialIndex}

   All new geofences added to the system will use the updated cell tower information. Run the following command to update all existing geofences:

   \texttt{rdl} > \texttt{reverseGeocode}

Deleting event stream subscriptions and geofences

Some cleanup is required when you delete event stream subscriptions (ESS) or geofences via the RDL.

1. Run the following RDL commands:
   
   \texttt{rdl} > \texttt{delete} --datatype Subscription [--file ...]
   \texttt{rdl} > \texttt{delete} --datatype Geofence [--file ...]

2. Run the rebuild command:
   
   \texttt{rdl} > \texttt{rebuildOptIns} --type ExOptInRebuildAll

   Alternatively, run the same command in RTISH:

   \texttt{rtish} > \texttt{rti refdata rebuildOptIns} --type ExOptInRebuildAll
Chapter 4: Loading Reference Data

**Repairing indexes**

You can repair indexes when the logs indicate that an error has occurred. You can run the commands as many times as needed.

1. Run the following RDL commands in the listed order:
   a. Fix the spatial index:
      ```
      rdl> buildspatialIndex
      ```
   b. Fix geofences and the associations between geofences and event subscriptions:
      ```
      rdl> reverseGeocode
      ```

2. Run the following RTISH command to clean old event subscriptions and optimize the runtime speed of the geofence index:

   ```
   rtish> rti refdata rebuildOptins --type LocationOptInRebuild
   ```

**Anonymizing reference data**

Configure the RTI system and utilities in a consistent way to leverage anonymization without breaking data consistency.

Anonymization is implemented using hashing. A cryptography one-way function is applied to the data, such as the IMSI. The anonymization is specified by two parameters:

- The algorithm used, such as:
  - MD5
  - SHA1
  - SHA256 (default)
- A salt value that makes the hash function system specific. Choosing a salt value should be performed by a security specialist.

Anonymization must be applied in the same way to both the events flowing from the probes and to the reference data loaded. Typically, RTI handles the anonymization, and you only have to configure the anonymization properties. Reference data does not have to be pre-processed before loading.

The following image displays an example anonymization flow:
RTI can anonymize any attribute of the data including some pre-defined attributes. For more information, refer to Anonymizing data.

**Configuring the reference data loader**

Configure the reference data loader according to the following system-wide values:

- rti.hasher.algorithm
- rti.salt

See [Common properties](#) for information on these properties.

**Note:** If you have already anonymized your reference data, you may configure the reference data loader as follows. Do not apply these settings to the grids.

- rti.anonymizer.noHashIMSI=true: If you have pre-anonymized any IMSIs using the same hash algorithm and salt as the global settings.
- rti.anonymizer.noHashMSISDN=true: If you have pre-anonymized any MSISDNs using the same hash algorithm and salt as the global settings.

**Configuring the ingester and ingest grid**

Configure settings to control the anonymization of incoming events from the probes. The settings are necessary to ensure that incoming events are anonymized properly.

Configure the Ingester and ingest grid according to the following system-wide values:

- rti.hasher.algorithm
- rti.salt

See [Common properties](#) for information on these properties.

**Note:** Change the anonymization settings for incoming events only after careful consideration and with security approval. If allowed, the following settings may be changed for testing purposes:
• **rti.anonymizer.noHashIMSI=true**: disables anonymization of IMSI data for incoming events. This setting should only be applied if the data stream has pre-anonymized IMSIs using the same hash algorithm and salt as the global settings. You can apply this switch for testing purposes only if reference data anonymization is also disabled. Ensure that un-anonymized events do not leave the system.

• **rti.anonymizer.noHashMSISDN=true**: MSISDNs are permitted to pass out of the system un-anonymized. This setting can only be used if allowed by security and privacy laws.

### Configuring external partner subscription setup anonymization in distribution grid

Configure settings for distribution grid components to control the anonymization of partner event stream subscriptions. The settings do not affect the anonymization of event data, which is handled by the ingest grid.

Configure the distribution grid according to the following system-wide values:

- **rti.hasher.algorithm**
- **rti.salt**

See [Common properties](#) for information on these properties.

**Note:** Change the anonymization settings for incoming events only after careful consideration and with security approval. If allowed, the following settings may be changed for testing purposes:

• **rti.anonymizer.noHashIMSI=true**: Requires partners to send pre-anonymized IMSIs when setting up subscriptions and opting in users. This setting should only be applied if the data stream has pre-anonymized IMSIs using the same hash algorithm and salt as the global settings. You can apply this switch for testing purposes only if reference data anonymization is also disabled. Ensure that un-anonymized events do not leave the system.

• **rti.anonymizer.noHashMSISDN=true**: MSISDNs are permitted to pass out of the system un-anonymized. This setting can only be used if allowed by security and privacy laws.

### Data loading process for event subscriptions

When you are setting up an RTI system to support event subscriptions, load data into the grids as follows:

1. Start the loader on the ingest grid.
2. Load the baseline reference data (cell towers, devices, and subscribers) into the ingest grid. For example:
   ```
   create --file CELL_OCT11.csv --datatype CellTower
   create --file TAC_DEVICE.csv --datatype Device
   ```
3. Load trace data into the ingest grid (optional). For example:
   ```
   create --file TRACE_DEVICE.csv --datatype Trace --tracetype Device
   create --file TRACE_LOCATION.csv --datatype Trace --tracetype Location
   ```
4. Load Global opt-in filters into the ingest grid. For example:

```bash
create --file OPTIN_LOCATIONS_A_ALL.csv --datatype LocationOptin
--optintype Global
create --file OPTIN_SUBSCRIBERS_A_62K.csv --datatype SubscriberOptin
--optintype Global
```

5. Start the loader on the distribution grid.

6. Load organization units into the distribution grid. For example:

```bash
create --file ORG_UNITS.csv --datatype Orgunit
```

7. Load opt-in filters into the distribution grid for the LANDS and LORS event streams. For example:

```bash
create --file OPTIN_LOCATIONS_A_ALL.csv --datatype LocationOptin
--optintype Location
create --file OPTIN_SUBSCRIBERS_A_62K.csv --datatype SubscriberOptin
--optintype Location
create --file OPTIN_LOCATIONS_A_ALL.csv --datatype LocationOptin
--optintype Transaction
create --file OPTIN_SUBSCRIBERS_A_62K.csv --datatype SubscriberOptin
--optintype Transaction
```

The system is ready to process network events and receive subscription requests using the Provisioning Manager. These subscriptions may apply additional "external" opt-in filters.

Most RDL loads do not need to be performed frequently. Depending on the volume and degree of change in the data, you can periodically apply updates or complete refreshes of the existing data.

Reference data types have different velocities (frequency of change) and volumes (expected number of records per load). The following table summarizes the velocity and volume for each data type as an operational guideline for maintaining the system. The RTI system does not impose limits or enforce any restrictions on these operations.
Table 6. Velocity and volume of reference data types

<table>
<thead>
<tr>
<th>Data type</th>
<th>Velocity</th>
<th>Volume</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cell towers</td>
<td>Daily (replacement in full)</td>
<td>Less than 100K records (total)</td>
</tr>
<tr>
<td>Devices</td>
<td>Monthly (replacement in full)</td>
<td>Less than 100K records (total)</td>
</tr>
<tr>
<td>LANDS opt-ins</td>
<td>Daily delta (update/delete), about 10% of overall volume</td>
<td>• Global: as sized</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• LANDS and LORS streams: as sized</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Blacklist: 2% of Global subscribers</td>
</tr>
<tr>
<td>Organization units</td>
<td>Weekly delta (update/delete)</td>
<td>Less than 100 records</td>
</tr>
<tr>
<td>Subscriber opt-ins</td>
<td>Daily delta (update/delete), about 10% of overall volume</td>
<td>• Global: as sized</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• LANDS and LORS streams: by default, 40% of Global subscribers</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Blacklist: 10% of Global subscribers</td>
</tr>
<tr>
<td>Subscriber profiles</td>
<td>Daily delta (update/delete), about 10% of overall volume</td>
<td>As sized</td>
</tr>
<tr>
<td>Trace</td>
<td>Daily delta (update/delete), about 10% of overall volume</td>
<td>Less than 1K for subscriber and device trace data; less than 100 for location trace data</td>
</tr>
</tbody>
</table>

**Rebuilding indexes and opt-ins**

After loading, refreshing, or updating reference data for cell towers, run the following maintenance operations:

1. Run the `buildSpatialIndex` command on the distribution grid.

   Cell towers are initially loaded onto the ingest grid for enrichment and filtering of events. Any time you update, delete, or load new cell tower data, rebuild the indexes on the distribution grid where spatial processing occurs.

2. Run the `rebuildOptins` command on the distribution grid.

   This command uses the spatial indexes in the previous command. The `rebuildOptins` command rebuilds both LANDS and subscriber opt-ins to facilitate location and subscription-based filtering of events. The rebuilding of LANDS opt-ins relies on the rebuilt indexes from the previous command. A list of network locations is built from the geofence information in the spatial indexes.

**fetchSubscriberMapping command**

Run the `fetchSubscriberMapping` command only if you are loading subscriber profile data and you are using the hashing mechanism to anonymize IMSI values during processing (`rti.anonymizer.noHashIMSI`). This command generates the subscriber profile mapping file, which returns the actual IMSI values that map to downstream hashed IMSI values.
RDL error messages

The following table lists the error messages returned by the RDL:

<table>
<thead>
<tr>
<th>Message</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exception reading from $filename</td>
<td>File could not be read. The exception stack trace will have the actual details.</td>
</tr>
<tr>
<td>Region named '{}' was not found</td>
<td>A GemFire storage region is missing in the cluster configuration. Consult product support.</td>
</tr>
<tr>
<td>Invalid DeviceTagger CSV row [ A,,B,C]. Sending error message to error channel.</td>
<td>Provided row is invalid. Details of the offending row are printed below this error message.</td>
</tr>
<tr>
<td>Invalid LinkGeoFence CSV row [ A,B,C,,,,K ]</td>
<td>Provided row is invalid. Details of the offending row are printed below this error message.</td>
</tr>
<tr>
<td>Invalid LocationOptIn CSV row [ A,,B,,C ]</td>
<td>Provided row is invalid. Details of the offending row are printed below this error message.</td>
</tr>
<tr>
<td>Invalid MobileEquipment CSV row [ A,B,,,C,,D]</td>
<td>Provided row is invalid. Details of the offending row are printed below this error message.</td>
</tr>
<tr>
<td>Invalid OrganizationalUnit CSV row [ A,B,,,,C ]</td>
<td>Provided row is invalid. Details of the offending row are printed below this error message.</td>
</tr>
<tr>
<td>Error occurred during the enrichment process.</td>
<td>Throws out the exception stack to log file plus the message.</td>
</tr>
<tr>
<td>Error occurred during the enrichment process.</td>
<td>Throws out the exception stack to log file plus the message.</td>
</tr>
<tr>
<td>Invalid SubscriberOptIn CSV row [ A,,B,,C ]</td>
<td>Provided row is invalid. Details of the offending row are printed below this error message.</td>
</tr>
<tr>
<td>Invalid SubscriberProfile CSV row [ A,,B,,C ]</td>
<td>Provided row is invalid. Details of the offending row are printed below this error message.</td>
</tr>
<tr>
<td>Invalid row. No valid subscriber key.</td>
<td>Provided row is invalid. Details of the offending row are printed below this error message.</td>
</tr>
<tr>
<td>Invalid SubscriberTagger CSV row [ A,,B,,C ]</td>
<td>Provided row is invalid. Details of the offending row are printed below this error message.</td>
</tr>
<tr>
<td>Invalid EventStreamSubscription CSV row [ A,,B,,C ]</td>
<td>Provides the invalid line and exception to log file.</td>
</tr>
</tbody>
</table>
### Chapter 4: Loading Reference Data

<table>
<thead>
<tr>
<th>Message</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unable to submit stream activate request.</td>
<td>Message put on to log file with exception stack traces.</td>
</tr>
<tr>
<td>Failed to process data on target cluster due to cluster member/s low memory. A detailed list of members affected by low memory.</td>
<td>Cannot insert records into cluster due to low memory. Call product support to increase number of cluster members.</td>
</tr>
<tr>
<td>Failed to process data on target cluster due to cluster shutdown.</td>
<td>Cluster has been shut down, resulting in RDL not being able to continue inserting records.</td>
</tr>
<tr>
<td>Key [ X ] does not meet serializability requirements</td>
<td>Key cannot be serialized and needs to be reviewed. Consult product support.</td>
</tr>
<tr>
<td>Timed out getting distributed lock for key X</td>
<td>This is an unexpected error. Consult product support.</td>
</tr>
<tr>
<td>Installed CacheWriter aborted the remove operation for key %s</td>
<td>An installed cachewriter on the storage region has aborted the delete operation. Consult product support.</td>
</tr>
<tr>
<td>Error during executeCRUDFunctionWithKeyValue function</td>
<td>Full exception stack trace sent to log file. Consult product support.</td>
</tr>
<tr>
<td>Error occurred during geofence subscription linking [Id=%s, memberId=%s, memberName=%s, status=%s]</td>
<td>There was an error attempting to link geofences to subscriptions. Details are provided in the log file.</td>
</tr>
<tr>
<td>Failure occurred during linking of geofences to subscription.</td>
<td>Log file message</td>
</tr>
<tr>
<td>Error executing function with {}</td>
<td>Log file message</td>
</tr>
<tr>
<td>Unable to submit Location optin rebuild trigger.</td>
<td>Log file message with full exception stack trace.</td>
</tr>
<tr>
<td>Unable to submit subscriber optin rebuild trigger.</td>
<td>Log file message with the exception raised.</td>
</tr>
<tr>
<td>SPATIAL: Unable to submit spatial index build trigger.</td>
<td>There was an error attempting to build a spatial index. Details are provided in the log file.</td>
</tr>
<tr>
<td>Title translation returned exception: {}</td>
<td>Log file message with full exception stack trace.</td>
</tr>
<tr>
<td>Error executing function with {}</td>
<td>Log file message with full exception stack trace.</td>
</tr>
<tr>
<td>Error extracting mapping</td>
<td>Log file message with full exception stack trace.</td>
</tr>
<tr>
<td>Error occurred while trying to delete an existing mapping file.</td>
<td>There was an error attempting to delete a file. Full details are provided in the log.</td>
</tr>
<tr>
<td>Error occurred during writing load status context process.</td>
<td>There was an error writing to a file. Full details are provided in the log.</td>
</tr>
<tr>
<td>Message</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------------------------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>ERROR: Unable to create resource. Check input file: filename.txt</td>
<td>Unable to create a file resource on the specified directory path. This may be due to incorrect permissions. Review and correct directory permissions to allow RDL to write back to disk.</td>
</tr>
<tr>
<td>ERROR: Not connected to cluster for data loading.</td>
<td>The RDL tool is unable to connect to the ingest cluster. Contact support.</td>
</tr>
<tr>
<td>ERROR: Invalid file extension: filename.txt</td>
<td>Provided filename has an unsupported file extension. Only files with either .gz or .csv are supported.</td>
</tr>
<tr>
<td>ERROR: Invalid data file: filename.csv</td>
<td>Provided filename is invalid for one of the following reasons: file not found, passed filename does not correspond to an actual file, or file does not have read permissions.</td>
</tr>
<tr>
<td>Completed subscriber profile mapping failed. Time(150). Failed to write results from fetch subscriber mapping process to file. Check log file for further details.</td>
<td>The subscriber profile mapping failed because the results could not be written to a local file. Refer to the RDL log file for further information.</td>
</tr>
<tr>
<td>Failed to write results from fetch subscriber mapping process to file.</td>
<td>There was an error writing to a file. Full details are provided in the log.</td>
</tr>
<tr>
<td>Invalid records placed in file [ filename.csv ]</td>
<td>There are invalid records that cannot be parsed for the specified operation.</td>
</tr>
<tr>
<td>Failed to create file for invalid csv records.</td>
<td>Unable to create a file on the specified directory path. This may be due to incorrect permissions. Review and correct directory permissions to allow RDL to write failed CSV rows to file.</td>
</tr>
</tbody>
</table>
This chapter presents the following topics:

- **Controlling how events flow** ................................................................. 65
- **Event flows through the distribution grid** ........................................... 69
- **Location based service provisioning** .................................................. 71
Controlling event flow

This section explains how events move through the RTI system based on opt-ins and other processing. It also explains how events output to transaction or location files and become available to external subscriptions.

RTI opens a TCP port for each protocol that is enabled in the system. When a message comes in on a port, it is parsed into a ProtocolEvent and sensitive data is anonymized. The system then applies a set of filters that stop processing if the event contains no IMSI, event time, or location set in the ProtocolEvent. You can enable or disable each filter independently. The system moves an unfiltered event into the protocol's specific GemFire region where the ingest grid picks it up and continues processing. To increase throughput, events are batched before they are put into the GemFire region. Configure the number of events in a batch using the rti.ingester.<protocol>.batch.size command.

To trace the event flow through this chain, browse the JMX beans in the io.pivotal.rti.ingester.<protocolName>.source package. The flow starts with the NettyTCPServerInboundChannelAdapter bean and ends with the output MessageChannel.

The ingest grid filters, enriches, and transforms event messages before pushing them to the distribution grid using the WAN gateway for the application of business logic.

The ingest grid consists of the following Spring Integration components:

- Transformers
- Filters
- Enrichers
- Output handlers to WAN and GemFire cache

The ingest grid Event Flow can be divided into four steps:

- Ingest grid flow invocation
- Enrich filter flow
- WAN gateway flow
- Tracing flow

When the event passes through the ingestion flow, it proceeds to the main ingest grid flow, which performs the following event processing:

1. Anonymizes the event by hashing all sensitive data:
   - International Mobile Station Equipment Identity (IMEI)
   - IMEI Software Version (IMEISV)
   - IMSI
   - Mobile Station International Subscriber Directory Number (MSISDN)
   - Target phone call digits
2. Checks whether the event needs to be tagged for tracing based on location tag, subscriber tag, or device tag GemFire region. If any one of the criteria is enabled, then the event is tagged for tracing.

3. If event is tagged for tracing, then the event is scrubbed to remove all sensitive data. The following data are all set to empty strings:
   - International Mobile Station Equipment Identity (IMEI)
   - IMSI
   - Mobile Station International Subscriber Directory Number (MSISDN)
   - Target phone call digits

4. Builds the message for further processing:
   - PAYLOAD: Anonymized event
   - HEADER:
     - x-rti-raw-length: The raw event data length
     - x-rti-raw-bytes: The raw event data bytes
     - x-rti-raw-byte: The raw event type
     - x-rti-region-name: The GemFire event region name
     - traceLog: true if the event must be traced

5. The anonymized event with the headers is published to the input channel.

![Diagram of Main Ingest Grid Flow](image)

**Figure 5. Main Ingest Grid Flow**

**Enrich filters flow**

Applying enrichment filters drops events with invalid data. This is a two-fold process:

1. Verify that the event includes the following data:
   - eventTime
   - IMSI
2. The CellTower filter checks the event against the CellTower GemFire region based on the cell key and passes only those events that originated from the corresponding set of cell towers.

The filters can be enabled or disabled individually based on configurable properties.

Following filtering, the ingest grid flow is as follows:

1. Blacklisting
2. Location OR subscriber optin
3. ClearNonHashedFields
4. EventToSubscriber

**Blacklisting**

The blacklist filter blocks events from further processing based on the IMSI or LAC if the subscribers or locations, respectively, are explicitly blacklisted. The list of blacklisted IMSI's and LAC's are pre-loaded as part of reference data loading step into the ingest grid regions:

- OPTIN_BLACKLIST_Locations
- OPTIN_BLACKLIST_Subscribers

This step involves checking for the following (OR) condition(s):

- Location blacklist: Check whether the LAC and cellular ID (Cell ID) associated to the event is opted in for location blacklist. The GemFire region of interest at this step is OPTIN_BLACKLIST_Locations.

- Subscriber blacklist: If the location blacklist filter is disabled (not true), then this filter condition is invoked to check whether the subscriber (IMSI) associated to the event has opted in for subscriber blacklist based on the LAC) and Cellular ID. The GemFire region of interest at this step is OPTIN_BLACKLIST_Subscribers.

The filters can be enabled or disabled individually based on the following configurable properties:

<table>
<thead>
<tr>
<th>Property name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>rti.ingestgrid.filters.blacklist.enabled</td>
<td>Disable blacklist filter</td>
</tr>
<tr>
<td>rti.ingestgrid.filters.blacklist.location.disabled</td>
<td>Disable location blacklist filter</td>
</tr>
<tr>
<td>rti.ingestgrid.filters.blacklist.subscriber.disabled</td>
<td>Disable subscriber blacklist filter</td>
</tr>
</tbody>
</table>
Location OR subscriber optin
Location or subscriber option filters events for further processing based on the IMSI or LAC if the subscribers or locations, respectively, are explicitly opted in. The list of opted-in IMSI’s and LAC’s is pre-loaded as part of reference data loading step into the ingest grid regions:

- OPTIN_GLOBAL_Locations
- OPTIN_GLOBAL_Subscribers

The filters can be enabled or disabled individually based on the following configurable properties:

Table 9. LORS filter properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rti.ingestgrid.filters.globalLocationOptin.enabled</td>
<td>Enable global location optin filter</td>
</tr>
<tr>
<td>rti.ingestgrid.filters.globalSubscriberOptin.enabled</td>
<td>Enable global subscriber optin filter</td>
</tr>
</tbody>
</table>

ClearNonHashedFields
All non-hashed event fields are scrubbed and cleared for further processing.

EventToSubscriber
The raw event object is transformed to a subscriber object and sent to the WAN gateway from the GemFire adapter.

Tracing flow
RTI supports tracing events in the ingest grid based on the following reference data:

- Location
- Subscriber
- User equipment

Traced events are tagged throughout the RTI system and are logged by each application in the system.
Device tracing

This flow sends all traced user equipment in tagged events to the USEREQUIPMENT_Trace GemFire Region if they aren't blacklisted Subscribers and Location events.

The reference data GemFire regions of interest are:
- OPTIN_BLACKLIST_Locations
- OPTIN_BLACKLIST_Subscribers

Device intercept tracing

This flow sends all intercepted user equipment in tagged events to the USEREQUIPMENT_Intercept GemFire region.

Subscribers intercept tracing

This flow sends all intercepted subscribers in tagged events to the SUBSCRIBERS_Trace GemFire region.

Event flows through the distribution grid

The two primary outputs of the distribution grid are derived by matching events against two combinations of opt-in applied to the stream. These combinations, the logical "or" and "and" of location and subscriber opt-in, are referred to as Location OR Subscriber (LORS) and Location AND Subscriber (LANDS).

This section covers the LORS and LANDS flows.

LANDS and LORS

The distribution grid determines whether events received from the ingest grid are eligible for LANDS and LORS processing.
LANDS and LORS flows can be configured individually to whitelist (default) or blacklist events. Whitelisted flows require events to match the specified criteria for further processing. That is, those events are *opted in*. Blacklisted flows block events that match the criteria from further processing. That is, those events are *not* opted in. Optin types:

- **Location optin**: An event is opted in when its location area code (LAC) matches a LAC (and optionally Cell ID and time window) loaded via the Reference Data Loader into the grid.

- **Subscriber optin**: An event must be opted in based on either its LAC or its Cell ID and for the required time interval. Subscriber optins are looked up based on the event's International Mobile Subscriber Identity (IMSI).

If an event is opted in for LORS, but LORS is configured for blacklist mode, then the event is not processed by the LORS flow. The distribution grid only permits routing if the event is not opted in for LORS. On the other hand, if the event is opted in for location or subscriber, and LORS is configured for whitelist mode, then the event is processed by LORS.

The same logic applies to the LANDS flow.

---

**Note:** Location optin and/or subscriber optin can be disabled. If disabled, the event is automatically considered opted in for that optin.

Opted in events are routed to LORS or LANDS processing. Events ineligible for either LORS or LANDS are discarded.

---

**Figure 7.** LORS and LANDS flow
LORS flow
If network analytics is configured, then the requisite analytics regarding the event are captured (see Managing metrics). The event is then made available for consumption by the client via file, AMQP exchange, JDBC based on the configuration.

LANDS flow
The event is made available for consumption by the client via file, AMQP exchange, JDBC based on the configuration.

Location-based service provisioning
The RTI location based services (LBS) rely on the Reference Data Loader (RDL) command line tool and the Provisioning Manager interface to load reference data and listen for Event Subscription Service (ESS) subscription requests, respectively.

The following diagram shows how LBS data flows into and out of the RTI system.

Reference data loader
The Reference Data Loader (RDL) is a Java application that functions as a client to the RTI grid servers. RDL provides a command-line shell for loading all types of reference data and running other maintenance operations. You load reference data that is used to qualify events for processing as they stream through the system using the RDL.

You can load large CSV files into GemFire regions for reference during transformation and enrichment of event data. As events stream into the Ingester and Distribution grids,
lookups match the events with criteria defined by filters. In turn, events are included in or excluded from further processing and distribution to consumers.

In general, the RTI system relies on the following types of core reference data:

- **Enrichment data**: baseline network information about cell towers, mobile devices, and subscribers. This information enriches raw ingested events by providing specific details about locations, equipment, and users.

- **Filtering data**: opt-ins for locations and subscribers. This information identifies consent to track certain users and locations, which in turn determines whether certain events qualify for distribution.

- **Subscriptions**: data in the form of event stream subscriptions, as requested by predefined organization units (OUs). This information defines the criteria for delivering event streams to third parties, including geofences and additional opt-in filters.

**Provisioning Manager**

The Provisioning Manager is a public interface that listens for event subscriptions from client applications. It is the go-between that connects consumer requests from the outside world to the processing that occurs within the RTI system. Consumers (client applications) ask to receive events that match given subscriber and geofence criteria by sending requests to the Provisioning Manager via the AMQP interface (RabbitMQ) in a dedicated request queue. These message requests travel through RabbitMQ as sets of rows.

The request queue and a corresponding response queue are created automatically for each OU when reference data for that OU is loaded successfully. Organization name and organization unit fields in the OU reference data determine the queue names. The OU can create any number of ESS subscriptions, define any number of geofences, and link any number of those geofences to any of its ESS subscriptions.

You need to create and share RabbitMQ credentials for the OU as well as share the API key generated by the RTI system following the reference data load, in order for the OU to connect to RabbitMQ. If the API key or credentials are wrong, then the OU client application cannot connect to the request or response queues. When the OU has successfully instructed the PM to set up an ESS, the PM creates a new RabbitMQ routing key for each defined ESS. The OU can then create a queue for that routing key and consume the event stream.

**Event stream subscriptions**

The OU must send each of the following messages to its PM request queue in order to produce a successful ESS subscription:

- **ESS SUBMIT**: Includes a subscription identifier. This identifier distinguishes one ESS from another and forms part of the ESS queue name. The ESS SUBMIT message can indicate whether an ESS should be enabled or disabled. The message can indicate a date/time window that the ESS should be active, and it can indicate the output format for ESS messages that are sent to the ESS queue.

- **GF (geofence) SUBMIT**: Contains a GF identifier and a geofence definition, such as a WKT polygon defining a location-based zone. Online tools can be used to visualize and generate these WKT polygons for debugging purposes.
- **LINKGEOFENCE (LGF) SUBMIT**: Associates named geofences with ESS subscription identifiers. These messages can be marked to replace or augment any existing LGF definition.

- **STREAM OPTIN (SSO) SUBMIT**: Associates actual IMSIs (or MSISDNs) with the ESS. Depending on RTI properties, these IMSIs or MSISDNs might need to be anonymized in accordance with the hashing logic being used. For MSISDNs, the translation must match the entry in the subscriber profile reference data.

When an ESS is defined, an event (not otherwise filtered by optins and quality of data) is sent to the OU’s ESS queue if the IMSI (or MSISDN) matches the ESS’s list of interesting IMSIs (or MSISDNs), and the cell ID location is within the geofence defined for the ESS. The ESS must be enabled and within its active date/time window.
Chapter 6 Subscribing to Events

This chapter presents the following topics:

Subscribing to events .................................................................75
Provisioning data model ..............................................................75
Anonymizing data .................................................................76
Setting up a new event subscription ...........................................77
Chapter 6: Subscribing to Events

Subscribing to events

The purpose of the RTI solution is to satisfy requests from client applications, submitted in the form of event subscriptions. An event subscription is a request to receive a stream of network events that meet certain criteria. When a subscription is in place, a dedicated queue is created and the qualifying set of events that process through the system is sent to the client application in the form of messages on that queue. The lifecycle of operations for event subscriptions is handled in AMQP message format, from the initial setup of the subscription itself to the output of messages consumed by external applications. The Provisioning Manager is the mechanism for setting up subscriptions in RTI, and RabbitMQ is the broker (the medium for exchanging messages with requesting applications).

For information about the output of a subscription, see Output format for event streams.

Provisioning data model

To set up an event subscription, you need to define and send a set of messages that provision the entities shown in green in the following diagram: the subscription itself, a geofence, and the subscribers who have opted in (given consent to be targeted). To form a contract between the RTI system and the consuming application, each subscription must be associated with a predefined organizational unit (OU), as loaded into the grids via a Reference Data Loader (RDL) command. See Loading reference data.

![Provisioning data model](image)

**Figure 9. Provisioning data model**

An OU is an entity that is entitled to access RTI via AMQP to request subscriptions for an application; an OU is the basic unit of subscription for the RTI system. Typically, there is a one-to-one relationship between an OU and a consumer of an event stream; however, RTI does not impose any limits on the number of provisioning sources per organization.
Anonymizing data

RTI has an enricher to anonymize any field in the Ingest and Distribution grids. By default, the Enricher is turned off. To enable the anonymize-any-field Enricher, set the `rti.transformers.anonymizeAnyField.enabled` property to true. When it is enabled, set the `rti.transformers.anonymizeAnyField.fields` property to a CSV list of fields that need to be anonymized.

The following example shows how to enable and anonymize `ingestGridField1`, `ingestGridField2` and `ingestGridFieldN`:

```bash
rti set property --instances ingestgrid-one@* --name rti.enrichers.anonymizeAnyField.enabled --value true
rti set property --instances ingestgrid-one@* --name rti.enrichers.anonymizeAnyField.fields --value ingestGridField1, ingestGridField2, ingestGridFieldN
```

The field anonymization capability scans and anonymizes the contents of the protocol attribute map, protocol details map, and the protocol event for any matches for each configured field.

The following example shows how to anonymize the "key" field. The key is a field in the ProtocolEvent domain object and could be a field in the protocol details map.

```bash
rti set property --instances ingestgrid-one@* --name rti.enrichers.anonymizeAnyField.fields --value key
```
Chapter 6: Subscribing to Events

Setting up a new event subscription

To set up an event subscription, follow these steps:

1. Ensure the RTI system is fully configured and the appropriate reference data is loaded from the network operator’s point of view.
2. Prepare a RabbitMQ client.
3. Configure the client (with connectivity to the RTI system).
4. Create a subscription by defining and submitting messages.

To submit provisioning messages to the RTI system, the subscribing application must set up a TCP/IP connection to one or more AMQP brokers (RabbitMQ). The application will need OSI Layer 4 connectivity to the RabbitMQ endpoints. The brokers may be in a clustered setup with multiple endpoints. In any case, bidirectional communication must be possible between the brokers and the subscribing application.

A simple way of getting started with the Provisioning Manager is to use a Groovy script client, as shown in the following example. Groovy is a programming language that executes on top of the JVM. You can download and install Groovy from http://www.groovy-lang.org/ to run the following example against your RTI system.

The script requires access to the RabbitMQ Client Library (http://www.rabbitmq.com/java-client.html) in its classpath. However, Groovy does not require any manual download and classpath manipulation. The @Grab annotation fetches the RabbitMQ library from the Maven Repository (http://mavenrepository.com/).

```
@Grab(group='com.rabbitmq', module='amqp-client', version='3.1.4')
import com.rabbitmq.client.*
import java.util.*
```

After the required imports, the following declarations summarize the connection metadata required by the client:

```
orgUnit = "BRN_DG"
apiKey = "6d051b96f732e6607b30f9b9d5caf39a"
exchange = "BRN.DG"
routingKey = "BRN.DG.request"
responseQName = "BRN.DG.response"
```
Chapter 6: Subscribing to Events

This example also introduces three helper methods described in the table below.

**Table 10. Example helper methods**

<table>
<thead>
<tr>
<th>Helper method</th>
<th>Purpose</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>newBasicProps</td>
<td>Build AMQP message properties and headers</td>
<td>If replyTo has been defined, a temporary/anonymous reply queue is used; the commented-out statement shows how to inject an arbitrary queue ID into the message envelope. The headers key-value structure is required to send the API key information to the Provisioning Manager. The owner of the RTI organization unit distributes the API key.</td>
</tr>
<tr>
<td>sendAndConsume</td>
<td>Send AMQP messages to the broker and block on the reply queue consumer</td>
<td>basicPublish is an AMQP method to send events over a lightweight connection called a channel. In addition to the target exchange, routing key, and properties, basicPublish requires the payload to be a byte array.</td>
</tr>
<tr>
<td>buildSSO</td>
<td>Aggregate an IMSI list with an event subscription ID, required as part of the subscription opt-in (SSO) specification</td>
<td>IMSIs starting with “#” will not be processed (they are treated as comments).</td>
</tr>
</tbody>
</table>

Here is the source code for the helper methods:

```python
def newBasicProps(contentType, replyTo, orgUnit, apiKey) {
    headers = {
        "decoder.apiKey": apiKey
    }
    msgId = UUID.randomUUID()
    basicPropsBuilder = new AMQP.BasicProperties.Builder()
    basicPropsBuilder.appId("APP01")
    basicPropsBuilder.userId(orgUnit)
    basicPropsBuilder.messageId(msgId.toString())
    basicPropsBuilder.expiration("60000") // one minute
    basicPropsBuilder.timestamp(new Date())
    basicPropsBuilder.correlationId("my_correlation_id")
    basicPropsBuilder.contentTypeId(contentType);
    basicPropsBuilder.contentEncoding("UTF-8")
    basicPropsBuilder.deliveryMode(2) // persistent
    // Un-comment and modify the following line to inject
    // a specific queue id into the envelope
    // basicPropsBuilder.replyTo("VMW.EP.response." + msgId)
    if (replyTo != null) {
        basicPropsBuilder.replyTo(replyDecOk.getQueue())
    }
    basicPropsBuilder.headers(headers)
    basicProps = basicPropsBuilder.build()
}
```
def sendAndConsume(channel, props, message, responseQ, consumer, exchange, routingKey) {
    contentEncoding = "UTF-8"
    channel.basicPublish(exchange, routingKey, props, message.getBytes(contentEncoding))
    delivery = consumer.nextDelivery();
    delivery.getProperties()
}

def buildSSO(imsis, subscId) {
    def message = '';
    imsis.eachLine { line ->
        if (line && !line.trim().startsWith('#')) {
            message += line.tokenize(',').join(',') + ',' + subscId + System.getProperty("line.separator");
        }
    }
    return message
}

To connect to the Provisioning Manager, you must provide the username, password, host, port, and vhost. (vhost is an advanced multi-tenant property, which can be left as "/").

username = System.getProperty("rabbit.username", orgUnit)
password = System.getProperty("rabbit.password", orgUnit)
vHost = System.getProperty("rabbit.vhost", "/")
host = System.getProperty("rabbit.host", "127.0.0.1")
port = Integer.getInteger("rabbit.port", 5672)

A com.rabbit.client.ConnectionFactory builds a new com.rabbit.client.Connection object based on the previously enumerated connection parameters. The RabbitMQ unit of work is a channel. It can be obtained from a connection and should be reused within a single thread. RabbitMQ encourages each client to declare its own artifacts: queues and exchanges. However, RTI automatically creates these artifacts up front unless a custom reply queue is used. A specific form of queue declaration (queueDeclarePassive) prevents the queue from being created if it is not present. The reference to the queue is necessary to combine a queuing consumer and the basicConsume operation. The client blocks on each expected response from the reply queue.

The line-separated list of IMSIs is used in conjunction with the buildSSO method to construct SSO binding messages.

factory = new ConnectionFactory()
factory.setUsername(username)
factory.setPassword(password)
factory.setVirtualHost(vHost)
factory.setHost(host)
factory.setPort(port)
connection = factory.newConnection()
Chapter 6: Subscribing to Events

```javascript
channel = connection.createChannel()
replyDecOk = channel.queueDeclare()
//replyTo = relyDecOk.getQueue()
replyTo = null
respDecOk = channel.queueDeclarePassive(responseQName)
responseQ = respDecOk.getQueue()
//responseQ = replyTo
consumer = new QueueingConsumer(channel);
channel.basicConsume(responseQ, true, consumer);
IMSIS=""
49268f953247fe52490baa4d758da31124a846733c72cb3a2c84e0880336dfa2
e532463ef293b165ea21be4906c75943d93593bd6a2f4c291d085b6aae71cb
fe33590a5e001700a2b99b105b681ce87614c6dd2fc96d716c2ee521bd879a"
```

Creating a subscription

To set up a geo-located feed for a subscription, publish at least four messages:

1. An event stream subscription (ESS) message: this creates an arbitrary network events feed
2. A geofence message: this defines an arbitrary region that is of interest to the consumer
3. A linkgeofence (LGF) message: this links the ESS and geofence. You can connect n geofences to a single feed to scope its area of interest.
4. A stream subscription opt-in (SSO) message: this connects target (opted in) IMSIs to the ESS feed, listening for events that involve those phone subscribers.

```javascript
///////// MAJ A
///////// MAJ A
props = newBasicProps("application/text+delimited;" +
   "messageType=PROV.01.ESS.SUBMIT;" +
   "compression=false",
   replyTo, orgUnit, apiKey)
message = "DGT02-MAJ-A,true," +
   "2012-11-23T00:00:00+0000," +
   "2013-12-01T23:59:59.999+0000"
sendAndConsume(channel, props, message, responseQ, consumer,
   exchange, routingKey)
//All encompassing geofence for 162
props = newBasicProps("application/text+delimited;" +
   "messageType=PROV.01.GF.SUBMIT;" +
   "compression=false",
   replyTo, orgUnit, apiKey)
message = "DGT02-GF-MAJ-A,4326,0," +
   "POLYGON ((-0.1100621103962973 51.61251265985425," +
   "-0.1488594053073289 51.57421392474571," +
   "-0.173837824000553 51.53877864066322," +
   "-0.1694012575544113 51.51398512060664," +
   "-0.119206742291465 51.49336508464171," +
```
This batch of messages includes the definition of the following geofence around North London:

![London geofence 1 example](image)

**Figure 10. London geofence 1 example**
The following messages define another subscription with a North East London geofence.

```java
props = newBasicProps("application/text+delimited;" +
    "messageType=PROV.01.ESS.SUBMIT;" +
    "compression=false",
    replyTo, orgUnit, apiKey)
message = "DGT02-MAJ-B,false," +
    "2012-11-23T00:00:00.000+0000," +
    "2013-12-01T23:59:59.999+0000"
sendAndConsume(channel, props, message, responseQ, consumer,
    exchange, routingKey)
```

// All encompassing geofence for 162
```java
props = newBasicProps("application/text+delimited;" +
    "messageType=PROV.01.GF.SUBMIT;" +
    "compression=false", replyTo, orgUnit, apiKey)
message = "DGT02-GF-MAJ-B,4326,0," +
    "POLYGON ((-0.08033828345673122 51.57208861329614," +
    " -0.08643491296618455 51.5486884609079," +
    " -0.09574535141757878 51.5293796833076," +
    " -0.09234863502284418 51.51034951493516," +
    " -0.07440701159402319 51.49078180757915," +
    " -0.03467004699585052 51.45648414519002," +
    " 0.08784209966345102 51.54242192010991," +
    " 0.08669574108460809 51.55893403906479," +
    " -0.01526089215138082 51.59523179425271," +
    " -0.08033828345673122 51.57208861329614))")
```

```java
sendAndConsume(channel, props, message, responseQ, consumer,
    exchange, routingKey)
```

```java
props = newBasicProps("application/text+delimited;" +
    "messageType=PROV.01.LGF.SUBMIT;" +
    "compression=false", replyTo, orgUnit, apiKey)
message = "DGT02-MAJ-B,DGT02-GF-MAJ-B"
sendAndConsume(channel, props, message, responseQ, consumer,
    exchange, routingKey)
```

```java
props = newBasicProps("application/text+delimited;" +
    "messageType=PROV.01.SSO.SUBMIT;" +
    "translate=false; compression=false",
    replyTo, orgUnit, apiKey)
message = buildSSO(IMSIS, "DGT02-MAJ-B")
sendAndConsume(channel, props, message, responseQ, consumer,
    exchange, routingKey)
```
Finally, you can link the previously defined geofences to target the union of those two areas.

```java
props = newBasicProps("application/text+delimited; " +
                     "messageType=PROV.01.ESS.SUBMIT;" +
                     "compression=false", replyTo,
                     orgUnit, apiKey)
message = "DGT02-MAJ-AB,false," +
          "2012-11-23T00:00:00.000+0000," +
          "2013-12-01T23:59:59.999+0000"
sendAndConsume(channel, props, message, responseQ, consumer,
                exchange, routingKey)
```

```java
props = newBasicProps("application/text+delimited; " +
                     "messageType=PROV.01.LGF.SUBMIT;" +
                     "compression=false", replyTo,
                     orgUnit, apiKey)
message = "DGT02-MAJ-AB,DGT02-GF-MAJ-A;DGT02-GF-MAJ-B"
sendAndConsume(channel, props, message, responseQ, consumer,
                exchange, routingKey)
```

```java
props = newBasicProps("application/text+delimited; " +
                     "messageType=PROV.01.SSO.SUBMIT;" +
                     "translate=false;compression=false",
                     replyTo, orgUnit, apiKey)
message = buildSSO(IMSIS, "DGT02-MAJ-AB")
sendAndConsume(channel, props, message, responseQ, consumer,
                exchange, routingKey)
```

```java
emptyMsg = new byte[0]
channel.close()
connection.close()
```
The following Groovy script enables you to set up event subscriptions in the Provisioning Manager, with RabbitMQ as the broker (the medium for exchanging messages with requesting applications).

```groovy
//@Grab(group='com.rabbitmq', module='amqp-client', version='3.1.4')
orgUnit = "BRN_DG"
apiKey = "6d051b96f732e6607b30f9b9d5caf39a"
exchange = "BRN.DG"
routingKey = "BRN.DG.request"
responseQName = "BRN.DG.response"

def newBasicProps(contentType, replyTo, orgUnit, apiKey) {
    headers=[
        "decoder.apiKey": apiKey
    ]

    msgId = UUID.randomUUID()

    basicPropsBuilder = new AMQP.BasicProperties.Builder()
    basicPropsBuilder.appId("APP01")
    basicPropsBuilder.userId(orgUnit)
    basicPropsBuilder.messageId(msgId.toString())
    basicPropsBuilder.expiration("60000") //one minute
    basicPropsBuilder.timestamp(new Date())
    basicPropsBuilder.correlationId("my_correlation_id")
    basicPropsBuilder.contentType(contentType);
    basicPropsBuilder.contentEncoding("UTF-8");
    basicPropsBuilder.deliveryMode(2) //persistent
    if (replyTo != null) {
        basicPropsBuilder.replyTo(replyTo)
    }
    basicPropsBuilder.headers(headers)
```
basicProps = basicPropsBuilder.build()
}

def sendAndConsume(channel, props, message, responseQ, consumer, exchange, routingKey) {
    contentEncoding = "UTF-8"
    channel.basicPublish(exchange, routingKey, props, message.getBytes(contentEncoding))
    delivery = consumer.nextDelivery();
    delivery.getProperties()
}

/* Configuring a Provisioning Manager Client **/

username = System.getProperty("rabbit.username", orgUnit)
password = System.getProperty("rabbit.password", orgUnit)
vHost = System.getProperty("rabbit.vhost", "/")
host = System.getProperty("rabbit.host", "127.0.0.1")
port = Integer.getInteger("rabbit.port", 5672)

factory = new ConnectionFactory()
factory.setUsername(username)
factory.setPassword(password)
factory.setVirtualHost(vHost)
factory.setHost(host)
factory.setPort(port)
connection = factory.newConnection()
channel = connection.createChannel()

if (responseQName == null) {
    replyDecOk = channel.queueDeclare()
    replyTo = replyDecOk.getQueue()
    responseQ = replyTo
} else {
    replyTo = responseQName
    respDecOk = channel.queueDeclarePassive(responseQName)
    responseQ = respDecOk.getQueue()
}
replyTo = null

consumer = new QueueingConsumer(channel);
channel.basicConsume(responseQ, true, consumer);

props = newBasicProps("application/text+delimited;" +
    "messageType=PROV.01.ESS.SUBMIT;" +
    "compression=false",
    replyTo, orgUnit, apiKey)

message = "SUB05,true,"
    "2012-11-23T00:00:00.000+0000,"
    "2013-12-01T23:59:59.999+0000"
sendAndConsume(channel, props, message, responseQ, consumer, exchange, routingKey)

props = newBasicProps("application/text+delimited;" +
    "messageType=PROV.01.GF.SUBMIT;" +
    "compression=false",
    replyTo, orgUnit, apiKey)
message = "MK01,4326,0,POLYGON((-0.88737 51.969212, " +
    "-0.88737 52.196331, " +
    "-0.59182 52.196331, " +
    "-0.59182 51.969212, " +
    "-0.88737 51.969212))"
sendAndConsume(channel, props, message, responseQ, consumer, exchange, routingKey)

props = newBasicProps("application/text+delimited;" +
    "messageType=PROV.01.LGF.SUBMIT;" +
    "compression=false",
    replyTo, orgUnit, apiKey)
message = "SUB05,MK01;LDN-M25,true"
sendAndConsume(channel, props, message, responseQ, consumer, exchange, routingKey)

props = newBasicProps("application/text+delimited;" +
    "messageType=PROV.01.SSO.SUBMIT;" +
    "translate=true;compression=false",
    replyTo, orgUnit, apiKey)
message = "447560166505,SUB05"
sendAndConsume(channel, props, message, responseQ, consumer, exchange, routingKey)

props = newBasicProps("application/text+delimited;" +
    "messageType=PROV.01.SST.SUBMIT;" +
    "compression=false",
    replyTo, orgUnit, apiKey)
message = "SUB05"
sendAndConsume(channel, props, message, responseQ, consumer, exchange, routingKey)

props = newBasicProps("application/text+delimited;" +
    "messageType=PROV.01.SSM.SUBMIT;" +
    "translate=true;compression=false",
    replyTo, orgUnit, apiKey)
message = "SUB05,true"
sendAndConsume(channel, props, message, responseQ, consumer, exchange, routingKey)

//message =
emptyMsg = new byte[0]

channel.close()
connection.close()
This chapter presents the following topics:

- **Ingesting events** ................................................................. 89
- **Enriching data** ................................................................ 90
- **Using selectable geocoding** ........................................... 93
- **Tracking geofence activity** ............................................. 97
Ingesting events

JSON (JavaScript Object Notation) is a lightweight data storage and exchange format. The JSON protocol adapter enables you to feed JSON messages to RTI over a TCP stream. The default port is 29009.

Incoming data must contain one message per line. Each line is a JSON object. An object is an unordered set of name/value pairs. The object can contain any properties listed in the RTI data dictionary. You can view the dictionary using `rti show dictionary` command. The data dictionary contains the description of each field as well as the expected format.

For example, the `msisdn` field:

- Description: Default phone number (user can also have other non-default ones).
- Format: String in E.164 format.

Note: The JSON message format is compatible with the Syphon output.

Example of JSON message:

Note: The example is formatted on multiple lines for display purposes. Actual messages should contain one line terminator at the end of each record.

```json
{
  "protocolName": "adr",
  "key": "d91b07d0-2492-11e4-0b2b-1c99e99ab165",
  "ingestTime": 1408117385421,
  "primaryId": "gloucester-imsi-5",
  "temporaryId": "09D3ED34",
  "eventStartTime": 1318528746,
  "eventEndTime": 1318529746,
  "protocolType": 2,
  "protocolDetails": {
    "dtapMMCause": 65535,
    "startTimeUSec": 91000,
    "dtapCCCause": 65535,
    "dtapRRCause": 65535,
    "endTimeUSec": 463000,
    "timeoutBits": 0,
    "callNumber": 0,
    "equipmentId": 0,
    "bssMapCause": 9,
    "bsc": 671089903,
    "conditionIndicator": 16384,
    "processorId": 0,
    "subscriberInfo": 0,
    "msc": 671092653,
    "callType": 9,
    "realImsi": "gloucester-imsi-5",
    "latitude": 90,
    "longitude": 180,
    "radii90": -1,
    "firstLac": 177,
  }
}
```
Run the following command to push a file containing JSON messages to the adapter:

```
nc 127.0.0.1 29009 < sample.dat
```

Adjust the IP address, port number and name of the file as needed.

**Enriching data**

**Dynamically enriching data with learning enricher**

RTI has the capability to learn from past events and enrich events from learned data. Be aware that enabling this feature may add some noticeable processing overhead when RTI writes values it sees for the first time. The overhead of subsequent enrichment by reading values and adding them to events is negligible.

RTI has implemented learning on the following fields:

- IMEI
- pairedMsip
- msipActive
- bearerId
- Location

If RTI encounters a value for one of these fields that it has not seen before, RTI will use that field value for future enrichment of events originating from the same Subscriber. Except for Location, RTI considers data to be new when an event has a field value that has changed from the previously learned value. Location differs by considering a Location with a better accuracy or if the learned Location times-out.

**Table 11. Learning enricher configuration properties**

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>io.pivotal.rti.t.data.si.enrichers.LearningEnricher.enabled</td>
<td>Set to true to enable the learning enricher.</td>
<td>false</td>
</tr>
<tr>
<td>io.pivotal.rti.t.data.si.enrichers.LearningEnricher.IMEI.enabled</td>
<td>Enable the IMEI learning on enricher.</td>
<td>false</td>
</tr>
<tr>
<td>io.pivotal.rti.t.data.si.enrichers.LearningEnricher.pairedMsip.enabled</td>
<td>Enable the pairedMsip learning on the enricher.</td>
<td>false</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>-------------------------------------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>----------</td>
</tr>
<tr>
<td>io.pivotal.rti.t.data.si.enrichers.LearningEnricher.msipActive.enabled</td>
<td>Enable the msipActive learning on the enricher.</td>
<td>false</td>
</tr>
<tr>
<td>io.pivotal.rti.t.data.si.enrichers.LearningEnricher.bearerId.enabled</td>
<td>Enable the bearerId learning on the enricher.</td>
<td>false</td>
</tr>
<tr>
<td>io.pivotal.rti.t.data.LocationTimeOut.timeout</td>
<td>The number of milliseconds before a Location times out. After it times-out, the enricher will save the next Location value received for subsequent enrichment.</td>
<td>120,000</td>
</tr>
<tr>
<td>io.pivotal.rti.t.data.si.enrichers.LearningEnricher.Location.enabled</td>
<td>Enable the Location learning on the enricher.</td>
<td>false</td>
</tr>
</tbody>
</table>

Examples for enabling the Learning Enricher service via RTISH:

```bash
rti set property --instances DistGrid1@* --name io.pivotal.rti.t.data.si.enrichers.LearningEnricher.enabled --value true
rti set property --instances DistGrid1@* --name io.pivotal.rti.t.data.si.enrichers.LearningEnricher.IMEI.enabled --value true
rti set property --instances DistGrid1@* --name io.pivotal.rti.t.data.si.enrichers.LearningEnricher.pairedMsip.enabled --value true
rti set property --instances DistGrid1@* --name io.pivotal.rti.t.data.si.enrichers.LearningEnricher.msipActive.enabled --value true
rti set property --instances DistGrid1@* --name io.pivotal.rti.t.data.si.enrichers.LearningEnricher.bearerId.enabled --value true
rti set property --instances DistGrid1@* --name io.pivotal.rti.t.data.LocationTimeOut.timeout --value 120000
rti set property --instances DistGrid1@* --name io.pivotal.rti.t.data.si.enrichers.LearningEnricher.Location.enabled --value true
```

Through JMX, each one of the learning enricher fields can be dynamically enabled or disabled. Along with toggling the active state the JMX interface can provide metrics on the performance of the learning enricher.

**Note:** Configuration changes made via JMX are not saved, and will revert once an instance is restarted. Make sure to apply changes permanently by changing the property values via RTISH.
Using a JMX GUI tool such as JVisualvm, you can navigate to the Learning Enricher MBeans on distribution grid instances as follows:

- Top Level MBean: io.pivotal.rti.distgrid.processor
- SI MBean Type: ”MessageHandler”
- Each Learning Enricher bean listed under MessageHandler has an SI ”handler” bean

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inputEnrichment$child.LearningEnricherStart</td>
<td>Start/Stop the LE Service. Contains aggregate</td>
</tr>
<tr>
<td></td>
<td>counters for all enabled fields</td>
</tr>
<tr>
<td>inputEnrichment$child.LearningEnricherBearerId</td>
<td>Start/Stop/view Metrics on BearerId enrichment</td>
</tr>
<tr>
<td>inputEnrichment$child.LearningEnricherIMEI</td>
<td>Start/Stop/view Metrics on IMEI enrichment</td>
</tr>
<tr>
<td>inputEnrichment$child.LearningEnricherLocation</td>
<td>Start/Stop/view Metrics on Location enrichment</td>
</tr>
<tr>
<td>inputEnrichment$child.LearningEnricherMsipActive</td>
<td>Start/Stop/view Metrics on msipActive enrichment</td>
</tr>
<tr>
<td>inputEnrichment$child.LearningEnricherPairedMsip</td>
<td>Start/Stop/view Metrics on pairedMsip enrichment</td>
</tr>
<tr>
<td>inputEnrichment$child.LearningEnricherEnd</td>
<td>Metrics only. At this phase the learning enricher stores any changes and clears the Learning Enricher metadata from the message</td>
</tr>
</tbody>
</table>

Viewing the learning enricher data can be done through the GemFire gfsh command. The dynamic data is held in a Map data structure. Below is an example query that will list all of the dynamic data for a given IMSI.

1. Launch GFSH from the "rti-t/tools/gfsh/bin" directory and connect to a distribution grid Locator:
   
   ```bash
   $ cd /data2/smithj/develop/rti-t/tools/gfsh/bin/
   $ ./gfsh
   Monitor and Manage GemFire
   gfsh>connect --locator=172.28.8.4[11334]
   Connecting to Locator at [host=172.28.8.4, port=11334]..
   Connecting to Manager at [host=nrt-w4.dh.greenplum.com, port=1099]..
   Successfully connected to: [host=nrt-w4.dh.greenplum.com, port=1099]
   Cluster-2 gfsh>
   ```
2. Execute the OQL Query to retrieve the learned values from the GemFire cache Region:

Cluster-2 gfsh>query --query="select distinct ddr.key, sub.getKey(), sub.value.getValue() from /DynamicDataRegion.entries ddr, ddr.value sub ORDER BY ddr.key"

Result : True
startCount : 0
endCount : 20
Rows : 4

<table>
<thead>
<tr>
<th>Key</th>
<th>getKey</th>
<th>getValue</th>
</tr>
</thead>
<tbody>
<tr>
<td>30dccd248953e9ec6c9b45646ed34050</td>
<td>IMEI</td>
<td>gloucestXXXXXXxi-1</td>
</tr>
<tr>
<td>30dccd248953e9ec6c9b45646ed34050</td>
<td>bearerId</td>
<td>2</td>
</tr>
<tr>
<td>b69e9faa34a2a8d0d8b9307ac20008ab</td>
<td>IMEI</td>
<td>gloucestXXXXXXxi-1</td>
</tr>
<tr>
<td>b69e9faa34a2a8d0d8b9307ac20008ab</td>
<td>bearerId</td>
<td>4</td>
</tr>
</tbody>
</table>

NEXT_STEP_NAME : END

To narrow the results, you can add a WHERE clause on any of the selected fields. For example, to narrow the results to a single IMSI, use the following:

Cluster-2 gfsh>query --query="select distinct ddr.key, sub.getKey(), sub.value.getValue() from /DynamicDataRegion.entries ddr, ddr.value sub WHERE ddr.key='30dccd248953e9ec6c9b45646ed34050' ORDER BY ddr.key"

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

<table>
<thead>
<tr>
<th>Key</th>
<th>getKey</th>
<th>getValue</th>
</tr>
</thead>
<tbody>
<tr>
<td>30dccd248953e9ec6c9b45646ed34050</td>
<td>IMEI</td>
<td>gloucestXXXXXXxi-1</td>
</tr>
<tr>
<td>30dccd248953e9ec6c9b45646ed34050</td>
<td>bearerId</td>
<td>2</td>
</tr>
</tbody>
</table>

NEXT_STEP_NAME : END

Using selectable geocoding

A Provisioning Manager user can send geofence messages along with a spatial index on a per-geofence basis. The geofences are updated and reverse geocoded as per the specified index. Indexes are validated and errors reported.

Apart from the default spatial index, a user can specify a second spatial index: BufferedRadii90 (BR90). The BR90 spatial index uses the radii90 value provided in each cell tower record as a buffer to calculate an envelope around the cell tower location. This envelope indicates the cell tower coverage. In comparison, the default spatial index does
not use any buffer, and its envelope is essentially a point. When a request for reverse geocoding comes with a geofence and BR90 is specified, any envelope that intersects the geofence boundary is returned as part of the set of cell towers.

A Provisioning Manager user can send geofence messages along with a spatial index on a per-geofence basis. The geofences are updated and reverse geocoded as per the specified index. Indexes are validated and errors reported. Similar behavior is supported when the user creates or updates geofences via the Reference Data Loader. When geofence records include a spatial index parameter, they are reverse geocoded using that index.

**Note:** Do not create or update geofences using RDL. These commands should be used when bootstrapping the system only. RDL should be used to load geofences to the distribution grid only.

When using the Provisioning Manager (PM), a user must submit a geofence message (PROV.01.GF.SUBMIT).

- The spatial index should be included as the last field in the geofence CSV payload. That is, there is one spatial index per geofence specified. As before, multiple geofences may be sent in a single message, each separated by "\n".
- The following names are valid: "BufferedRadii90", "BR90", "Default Index". The names are not case-sensitive. If no name is specified, then the default index is used.
- If an invalid index is specified, then the geofence is not updated and the user is provided a response to indicate that the reverse geocoding could not be done. Also, the geofence is not added or updated.
- When a geofence is deleted, the remaining geofences in the system are reverse geocoded using the spatial index associated with their record.

**Example**

MK01,4326,0,POLYGON ((-0.88737 51.969212, .... -0.88737 51.969212)),BufferedRadii90
MK02,4326,0,POLYGON ((-0.414058195390 51.459122310700, .... -0.414058195390 51.459122310700))
MK03,4326,0,POLYGON ((-3.002792398970 51.616249432000, .... -3.002792398970 51.616249432000)),BR90

When geofences are loaded using the Reference Data Loader (RDL) (create/update), geofence records are read from a CSV file. A spatial index parameter is specified as part of the geofence record.

- The spatial index should be included as the last field in the geofence CSV payload. That is, there is one spatial index per geofence specified. As before, multiple geofences may be in the file, each record on a separate line.
- The following names are valid: "BufferedRadii90", "BR90", "Default Index". The names are not case-sensitive. If no name is specified, then the default index is used.
- If an invalid index is specified, then the geofence is not created or updated and the user is provided a response to indicate the invalid record identifiers that are placed in a file. The geofence is not added or updated, and reverse geocoding is not performed.

- When a geofence is deleted, the remaining geofences in the system are reverse geocoded using the spatial index associated with their record.

- JMX operations continue to use the default index only.

Example

VMW,EP,VMW_EP_61882_1,4326,0,POLYGON ((-3.00279239897051.616249432000,
-3.00779239897051.621249432000, -3.01279239897051.616249432000,
-3.00779239897051.611249432000, -3.00279239897051.616249432000)),BufferedRadii90

VMW,EP,VMW_EP_2587_2,4326,0,POLYGON ((-0.48794945309052.198830585300,
-0.49294945309052.203830585300, -0.49794945309052.198830585300,
-0.49294945309052.193830585300, -0.48794945309052.198830585300)),BR90

VMW,EP,VMW_EP_10537_3,4326,0,POLYGON ((-4.33772536955050.979301877000,
-4.34272536955050.984301877000, -4.34772536955050.979301877000,
-4.34272536955050.974301877000, -4.33772536955050.979301877000)),BR90

JMX operations

You can reverse geocode a geofence or a latitude and longitude pair using a JMX client such as VisualVM.

1. Install VisualVM, and configure the tool to monitor the RTI system. For instructions, see Using VisualVM for monitoring.

2. Connect to the distribution grid component and navigate to the SpatialOperationsManagement MBean using the MBean browser.

3. Expand the selection and click on the spatialOperations node.
Chapter 7: Enhancing Event Data

4. Submit one of the following queries using the query form:

- Query for all cell towers associated with a latitude/longitude pair. Enter the coordinates in the form along with the additional parameters. A list of cell towers associated with this geofence is returned, if present.

- Query for all cell towers associated with a geofence key. Enter the geofence key in the form along with additional parameters (described below). A list of cell towers associated with this geofence is returned, if present.
Additional parameters include:

- **useExistingIndex**

  _Note:_ Applies only when querying by geofence key.

  - Accepted values: true/false
  - Default value: true

    If true, then the spatial index name already associated with the geofence in the system is used to reverse geocode, and the value in the `spatialIndexName` parameter is ignored.

    If false or null, then the `spatialIndexName` value is used.

    This parameter is useful when determining the cell towers if the spatial index used were different from the one associated with the geofence. When querying by latitude and longitude the `spatialIndexName` parameter is always considered.

- **spatialIndexName**

  - Accepted value: The name of the spatial index.
    - BufferedRadii90
    - BR90
    - Default Index
  - Default value: Null

    If left empty, the default index is used. When querying by geofence key, the spatial index name is considered only if the `useExistingIndex` parameter value is false or empty. When querying by latitude/longitude pair, the spatial index name is always considered.

- **useCache**

  - Accepted values: true/false
  - Default value: true

    If true, then the system returns the cached response to the query.

**Tracking geofence activity**

The distribution grid can match which events belong to a set of geofences. When there is a geofence match, the geofence keys are added to the ProtocolDetailMap. The ProtocolDetailMap keys to look up to enable this content in an output formatter are:

- **entryGeoFences:** The list of geofences that the current event has just entered into.
- **exitGeoFences:** The list of geofences that the current event has just exited.
- **containedGeoFences:** A list of geofences that the current event is contained within.
To enable this behavior the following properties need to be set:

**Table 13. Geofence configuration properties**

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>rti.distgrid.enrichers.geofenceid.contains.enabled</td>
<td>Enable the enrichment of geofence data.</td>
<td>false</td>
</tr>
<tr>
<td>rti.distgrid.enrichers.geofenceid.entryexit.enabled</td>
<td>Enable the maintaining of state to know if a given event is an entry or exit event.</td>
<td>false</td>
</tr>
<tr>
<td>rti.distgrid.enrichers.geofenceid.entryexit.timeout</td>
<td>The number of seconds to maintain state before the state becomes stale and must be discarded.</td>
<td>300</td>
</tr>
<tr>
<td>rti.distgrid.enrichers.geofenceid.formatter.enabled</td>
<td>The default format is a collection of geofence keys. The formatter changes the output to be text.</td>
<td>False</td>
</tr>
<tr>
<td>rti.distgrid.enrichers.geofenceid.formatter.delimiter</td>
<td>The delimiter to use in the formatted output.</td>
<td>Semicolon character (;)</td>
</tr>
</tbody>
</table>

**Example**

The following example shows how to use the GeoFence entry, exit, and contains functionality to create a simple alerting capability using the LORS Where filter. A pair of GeoFences are set up and an event is tracked that moves into one geofence then the other.

Figure 14 shows the geofences indicated by the yellow boxes.
Figure 14. GeoFences

Steps to create the Geofence alerting capability

Follow these steps to create an alerting function using the LORS Where filter:

1. Enable the GeoFence entry, exit, and contains capability for all of the distribution grid members:

   rtish> rti set property --instances distgrid-one@* --name rti.distgrid.enrichers.geofenceid.contains.enabled --value true
   rtish> rti set property --instances distgrid-one@* --name rti.distgrid.enrichers.geofenceid.entryexit.enabled --value true
   rtish> rti set property --instances distgrid-one@* --name rti.distgrid.enrichers.geofenceid.formatter.enabled --value true
   rtish> rti set property --instances distgrid-one@* --name rti.distgrid.enrichers.geofenceid.formatter.delimiter --value |

2. Start or restart RTI so the enablement will take place.

3. Then using the RTISH command, connect to the distribution grid so you can make changes to that system.

   rtish> rti system connect --locator dist-grid-locator@*

4. Ensure that you have cell towers loaded so you can make GeoFence calculations.

   rtish> rti refdata celltower upsert --file CELL_TOWERS.csv

5. GeoFences have to be assigned to an organization unit so you must add at least one organization unit to the system before you add a GeoFence.

   rtish> rti refdata orgunit upsert --file orgunit.csv
where orgunit.csv contains:

gee_org,gee_ou,2012-11-23T00:00:00.000+0000,2099-12-01T23:59:59.999+0000

6. Add the desired GeoFences.

rtish> rti refdata geofence upsert --file geofences.csv

where geofences.csv contains:

gee_org,gee_ou,gee_gf-01,,,POLYGON ((-0.8879000393681502 52.24386255819387, -0.6636847526804746 52.21685900199603, -1.68834250963665 51.02141833672068, -1.974684963243017 51.03871923677913, -0.8879000393681502 52.24386255819387 )])
gee_org,gee_ou,gee_gf-02,,,POLYGON ((0.06549248381479522 51.53577992139218, -0.09596238063392715 51.42081748152997, -1.016274019869816 52.18054507123301, -0.8218663815045313 52.2882546786802, 0.06549248381479522 51.53577992139218 )])

7. After you add GeoFences to the system you have to rebuild RTI's spatial data indexes.

rtish> rti refdata buildspatialIndex

8. Once the spatial indexes are rebuilt you reverse geocode the GeoFence to the existing cell towers.

rtish> rti refdata geofence reversegeocode

9. You can deploy the stream and begin processing new GeoFence entries emitted by RabbitMQ:

rti stream deploy --definition geoentry.json

where geoentry.json is defined as:

```
{
    "name" : "geo_entry_alert",
    "enabled" : true,
    "where" : "protocolDetailMap['entryGeoFences'] != null",
    "select" : "imsi,latitude,longitude,entryGeoFences,exitGeoFences,containedGeoFences,
    cellId,lac",
    "outputType" : "amqp",
    "amqpExchange" : "geoentry.exchange",
    "amqpRoutingKey" : "geoentry.routing.key",
    "amqpBatchSize" : 1,
    "amqpBatchTimeout" : 100
}
```

The “where” clause used above states that whenever there is an entry GeoFence defined in the RTI event, a RabbitMQ message is to be emitted on the “geoentry.exchange” exchange using the routing key
“geoentry.routing.key”. The payload of the message sent on RabbitMQ is the csv content as specified in the “select” statement.

The following sample output shows when the events flow into GeoFence #1 and then into GeoFence #2:

```
13dc5,51.0851773184,-1.7722541319,gee_org_gee_ou_gee_gf-01,,gee_org_gee_ou_gee_gf-01,512,256
13dc5,51.5173461246,-0.1153961748,gee_org_gee_ou_gee_gf-02,gee_org_gee_ou_gee_gf-01,gee_org_gee_ou_gee_gf-02,1025,128
```
This chapter presents the following topics:

- **Using metrics and analyzing events** ................................................................. 103
- **Managing event analysis** .................................................................................. 103
- **Managing metrics** ............................................................................................ 112
- **Creating user-defined metrics streams** ......................................................... 133
- **KPI Store Function** .......................................................................................... 137
Using metrics and analyzing events

This feature enables you to gather actionable operations insights into data streams in a timely manner. The RTI system can track and calculate metrics along a number of dimensions to allow you to make meaningful business decisions. How many 3G (or 2G) voice and data messages are succeeding (or failing), by subscriber group or cell tower? How much bandwidth is used? The metrics calculated by the RTI system provide this type of analytics data.

Messages are analyzed and flagged in the distribution grid. The metrics engine sums up those flags and calculates derived metrics from them in the distribution grid.

Managing event analysis

RTI provides out-of-the-box analytics and metrics on event streams. Analytics flags are used to enrich events with information based on event data. These flags are subsequently used to calculate metrics. Some metrics have the same name as the analytics flag and in that case, the metric is usually a counter for that flag. Analytics flags are available in the ProtocolAttributesMap and metrics are available in the Metrics Output Stream.

Enabling event analysis

Enable network analytics on the distribution grid:

```
rti set property --instances <DistributionGridName>/*@*
--name rti.distgrid.lors.analytics.enabled --value true
```

**Note:** The `rti.distgrid.lors.analytics.enabled` switch enables event analysis, making the high-level flags available on the event stream. The flags are also necessary for the metrics stream engines.

Event analysis flags

The following event analysis flags are available to calculate metrics according to your proprietary expressions.

**Note:** Availability depends on the data sources and the associated content in the deployed environment.

Data types

Analytics flags are boolean or long (counts) values. Time-based flags are also represented as long values.

- 2G Call Setup
- 2G Dropped Call
- Bearer Data Attach
- Bearer Default/Dedicated Set-Up Time
- Bearer Rejections
- EPS Attach
- GnGi Network Bandwidth
Chapter 8: Using Metrics and Analyzing Events

- Packets
- PDN Connectivity
- PDP Context
- Radius Data Usage
- Radius Session Summary
- Radius Session Terminate
- Retransmission
- Session Drop
- Session Setup
- SMS 2G
- SMS 3G
- Tracking Area Update
- Upload/Download Speeds

2G call setup

- call.setup.2g.state
- call.setup.2g.state.css.transition.time
- call.setup.2g.state.csf.transition.time
- call.setup.mo.2g.fail.user
- call.setup.mo.2g.fail.network
- call.setup.mo.2g.fail.cause
- call.setup.mo.2g.success
- call.setup.mt.2g.fail.user
- call.setup.mt.2g.fail.network
- call.setup.mt.2g.fail.cause
- call.setup.mt.2g.success
- call.setup.2g.type.mo
- call.setup.2g.type.mt
- call.setup.2g.event.ssp
- call.setup.2g.event.cisp
- call.setup.2g.event.tosp
- call.setup.2g.event.ctmosp
- call.setup.2g.event.ctmtsp
- call.setup.2g.event.sss
- call.setup.2g.event.ciss
- call.setup.2g.event.toss
• call.setup.2g.event.ctss
• call.setup.2g.event.ssf
• call.setup.2g.event.ssuf
• call.setup.2g.event.ssnf
• call.setup.2g.event.cisf
• call.setup.2g.event.tosf
• call.setup.2g.event.tosuf
• call.setup.2g.event.tosnf
• call.setup.2g.event.ctsf
• call.setup.2g.event.tisf

2G dropped call
• call.dropped.2g.state
• call.dropped.2g.state.transition.time
• call.dropped.mo.2g.fail.user
• call.dropped.mo.2g.fail.network
• call.dropped.mo.2g.fail.cause
• call.dropped.mo.2g.success
• call.dropped.mt.2g.fail.user
• call.dropped.mt.2g.fail.network
• call.dropped.mt.2g.fail.cause
• call.dropped.mt.2g.success
• call.dropped.2g.type.mo
• call.dropped.2g.type.mt
• call.dropped.2g.event.rsf
• call.dropped.2g.event.cisf
• call.dropped.2g.event.rss
• call.dropped.2g.event.sss
• call.dropped.2g.event.ciss
• call.dropped.2g.event.cirs
• call.dropped.2g.event.toss
• call.dropped.2g.event.tosf
• call.dropped.2g.event.tors
• call.dropped.2g.event.ctss
• call.dropped.2g.event.ctsf
Chapter 8: Using Metrics and Analyzing Events

- call.dropped.2g.event.torf
- call.dropped.2g.event.ctmo
- call.dropped.2g.event.ctmt

**Bearer data attach**
- bearer.data.2g.attempt
- bearer.data.2g.fail.user
- bearer.data.2g.fail.network
- bearer.data.2g.fail.cause
- bearer.data.2g.success
- bearer.data.3g.attempt
- bearer.data.3g.fail.user
- bearer.data.3g.fail.network
- bearer.data.3g.fail.cause
- bearer.data.3g.success

**Bearer default/dedicated set-up time**
- bearer.dedicated.setup.time
- bearer.default.setup.time
- bearer.dedicated.setup.success
- bearer.default.setup.success

**Bearer rejections**
- bearer.setup
- bearer.setup.rejection
- bearer.setup.rejection.cause

**EPS attach**
- eps.attach.attempt
- eps.attach.fail.user
- eps.attach.fail.network
- eps.attach.fail.cause
- eps.attach.success
- eps.detach.fail
- eps.detach.fail.cause
- eps.detach.success
- eps.default.bearer.activate.attempt
- eps.default.bearer.activate.fail
Chapter 8: Using Metrics and Analyzing Events

- eps.default.bearer.activate.fail.cause
- eps.default.bearer.activate.success
- eps.dedicated.bearer.activate.attempt
- eps.dedicated.bearer.activate.fail
- eps.dedicated.bearer.activate.fail.cause
- eps.dedicated.bearer.activate.success

**GnGi network bandwidth**
- network.bandwidth.negotiated
- network.bandwidth.requested

**Packets**
- packets.uplink
- packets.downlink
- packets.unknown

**PDN connectivity**
- pdn.connect.fail.user
- pdn.connect.fail.network
- pdn.connect.fail.cause
- pdn.connect.success

**PDP context**
- pdp.context.2g.attempt
- pdp.context.2g.fail.user
- pdp.context.2g.fail.network
- pdp.context.2g.fail.cause
- pdp.context.2g.success
- pdp.context.3g.attempt
- pdp.context.3g.fail.user
- pdp.context.3g.fail.network
- pdp.context.3g.fail.cause
- pdp.context.3g.success

**Radius data usage**
- data.usage.octets.upload.cumm
- data.usage.octets.download.cumm
- data.usage.octets.upload.delta
- data.usage.octets.download.delta
Chapter 8: Using Metrics and Analyzing Events

- data.usage.session.time

**Radius session summary**
- session.summary.device.4g
- session.summary.device.3g
- session.summary.device.2g
- session.summary.device.unknown
- session.summary.rat.4g
- session.summary.rat.3g
- session.summary.rat.2g
- session.summary.rat.unknown
- session.summary.cell.4g
- session.summary.cell.3g
- session.summary.cell.2g
- session.summary.cell.unknown
- session.summary.device.4g.rat.4g
- session.summary.device.3g.rat.3g
- session.summary.device.2g.rat.2g
- session.summary.device.4g.not.rat.4g
- session.summary.device.3g.not.rat.3g
- session.summary.device.4g.not.rat.4g.cell.4g
- session.summary.device.4g.not.cell.4g
- session.summary.device.4g.not.rat.4g.cell.unknown
- session.summary.device.3g.not.rat.3g.cell.3g
- session.summary.device.3g.not.cell.3g
- session.summary.device.3g.not.rat.3g.cell.unknown

**Radius session terminate**
- session.terminate.success
- session.terminate.fail.user
- session.terminate.fail.network

**Retransmission**
- retransmission.time
- retransmission.packets.uplink
- retransmission.packets.downlink
- Session Drop
Chapter 8: Using Metrics and Analyzing Events

- session.drop.fail
- session.drop.status.fail
- session.drop.fail.cause
- session.drop.success

**Session setup**
- session.setup.fail
- session.setup.fail.cause
- session.setup.success
- session.setup.time

**SMS 2G**

**Notes:**

1. The following flags have been updated starting with release 1.0.0. from boolean fields to long values that represent a count:
   - "sms.mt.2g.attempt"
   - "sms.mo.2g.attempt"
   - "sms.mt.2g.success"
   - "sms.mo.2g.success"
   - "sms.mt.2g.fail.user"
   - "sms.mo.2g.fail.user"
   - "sms.mt.2g.fail.network"
   - "sms.mo.2g.fail.network".
   The corresponding SMS 2G metrics flags have also been updated to take into account these changes.

2. The *.timestamp flags allow for time correlation of the events. These flags could contain one or more timestamp values separated by the “+” sign, which correspond to the number of attempts, success or failures of the corresponding count flags.

3. The RP and CP *.cause flags allow cause analysis of the events. These flags could contain one or more error cause values separated by the “+” sign, which in turn match up with the timestamp values provided in the corresponding timestamp flags.

- SMS 2G submission:
  - sms.mo.2g.attempt
  - sms.mo.2g.attempt.timestamp
  - sms.mo.2g.cp.network.fail.cause
  - sms.mo.2g.cp.user.fail.cause
  - sms.mo.2g.fail.cause
Chapter 8: Using Metrics and Analyzing Events

- sms.mo.2g.fail.network
- sms.mo.2g.fail.network.timestamp
- sms.mo.2g.fail.user
- sms.mo.2g.fail.user.timestamp
- sms.mo.2g.rp.network.fail.cause
- sms.mo.2g.rp.user.fail.cause

sms.mo.2g.success SMS 2G delivery:

- sms.mt.2g.attempt
- sms.mt.2g.attempt.timestamp
- sms.mt.2g.cp.network.fail.cause
- sms.mt.2g.cp.user.fail.cause
- sms.mt.2g.fail.cause
- sms.mt.2g.fail.network
- sms.mt.2g.fail.network.timestamp
- sms.mt.2g.fail.user
- sms.mt.2g.fail.user.timestamp
- sms.mt.2g.rp.network.fail.cause
- sms.mt.2g.rp.user.fail.cause
- sms.mt.2g.success

**SMS 3G**

**Notes:**

1. The following flags have been updated from boolean fields to long values that represent a count:
   - sms.mt.3g.attempt
   - sms.mt.3g.success
   - sms.mt.3g.fail.user
   - sms.mt.3g.fail.network
   - sms.mo.3g.attempt
   - sms.mo.3g.success
   - sms.mo.3g.fail.user
   - sms.mo.3g.fail.network

2. The *.timestamp flags allow for time correlation of the events. These flags could contain one or more timestamp values separated by the “+” sign, which correspond to the number of attempts, successes, or failures of the corresponding count flags.
3. The RP and CP *.cause flags allow cause analysis of the events. These flags could contain one or more error cause values separated by the “+” sign, which, in turn, match up with the timestamp values provided in the timestamp flags.

- **SMS 3G submission:**
  - sms.mo.3g.attempt
  - sms.mo.3g.attempt.timestamp
  - sms.mo.3g.cp.network.fail.cause
  - sms.mo.3g.cp.user.fail.cause
  - sms.mo.3g.fail.cause
  - sms.mo.3g.fail.network
  - sms.mo.3g.fail.network.timestamp
  - sms.mo.3g.fail.user
  - sms.mo.3g.fail.user.timestamp
  - sms.mo.3g.rp.network.fail.cause
  - sms.mo.3g.rp.user.fail.cause
  - sms.mo.3g.success

- **SMS 3G delivery:**
  - sms.mt.3g.attempt
  - sms.mt.3g.attempt.timestamp
  - sms.mt.3g.cp.network.fail.cause
  - sms.mt.3g.cp.user.fail.cause
  - sms.mt.3g.fail.cause
  - sms.mt.3g.fail.network
  - sms.mt.3g.fail.network.timestamp
  - sms.mt.3g.fail.user
  - sms.mt.3g.fail.user.timestamp
  - sms.mt.3g.rp.network.fail.cause
  - sms.mt.3g.rp.user.fail.cause
  - sms.mt.3g.success

**Tracking Area Update**

- tau.generic.event
- tau.periodic.fail.user
- tau.periodic.fail.network
- tau.periodic.fail.cause
- tau.periodic.success
Chapter 8: Using Metrics and Analyzing Events

- tau.nonperiodic.fail.user
- tau.nonperiodic.fail.network
- tau.nonperiodic.fail.cause
- tau.nonperiodic.success
- tau.combined.fail.user
- tau.combined.fail.network
- tau.combined.fail.cause
- tau.combined.success
- tau.combined.attach.fail.user
- tau.combined.attach.fail.network
- tau.combined.attach.fail.cause
- tau.combined.attach.success

Upload/Download Speeds
- upload.download.requested
- upload.download.negotiated

Managing metrics

Metrics streams

The RTI system supports the following metrics streams, all of which have the same basic configuration options:

- **celltower**: Celltower metrics. Can also be configured to compute metrics per LAC (location area code).
- **device**: Device type metrics such as manufacturer, model, or model version.
- **subscribergroup**: Subscriber group metrics. For example, subscribers for a corporate account.
- **mobilenetwork**: Subscriber home network metrics per country code or network code.

A metrics stream is a rollup of one or more values along this dimension, such as aggregated dropped calls for each cell tower.

Calculated metrics consist of:

- Metric name, such as `uploaded-data`
- Metric value, such as `456789` (number of uploaded bytes)

Metrics have an associated time period, called a *time bucket*. Metrics are retained and calculated per time bucket. The RTI system outputs metrics data to a file on a regular interval. Once written, a file can be processed immediately. New files for a given time bucket replace for older data files for the same time bucket. You can remove any
The system retains metrics data for the current time, plus a configurable number of time buckets in the past and future, to handle events that arrive out of time.

**Enabling metrics**

The instructions in this section describe how to enable network analytics and capture metrics data.

1. Enable metrics and network analytics on the distribution grid:

   ```
   rti set property --instances <DistributionGridName>@* --name rti.distgrid.lors.metrics.enabled --value true
   ```

2. Enable event analysis. See Enabling event analysis for instructions.

3. Set additional properties using the same format:

   ```
   rti set property --instances <DistributionGridName>@* --name <PropertyName> --value <Value>
   ```

   See Metrics properties for the complete list of available metrics configuration properties.

**Available metrics**

This section lists metrics components available for inclusion in metrics streams.

**Note:** Availability depends on the data sources and the associated content in the deployed environment.

**Data types**

Rate and ratio are doubles. All other metric components are a long data type.

- 2G Call Setup
- 2G Dropped Call
- Bearer Data Attach
- Bearer Default/Dedicated Set-Up Time
- Bearer Rejections
- EPS Attach
- Gb
- GnGi
- GnGi Network Bandwidth
- IuPS
- Packets
- PDN Connectivity
- Radius Data Usage
- Radius Session Summary
- Radius Session Terminate
Chapter 8: Using Metrics and Analyzing Events

- Retransmission
- S1-MME Default/Dedicated Bearer Activate
- S1-MME Default/Dedicated Bearer Detach
- Session Drop
- SMS 2G
- SMS 3G
- SMS Combined
- Tracking Area Update
- Upload/Download Speeds
- XDATA

2G call setup

- call.setup.mo.2g.fail.user
- call.setup.mo.2g.fail.network
- call.setup.mo.2g.fail
- call.setup.mo.2g.success
- call.setup.mo.2g.attempts
- call.setup.mo.2g.fail.ratio
- call.setup.mo.2g.success.ratio
- call.setup.mt.2g.fail.user
- call.setup.mt.2g.fail.network
- call.setup.mt.2g.fail
- call.setup.mt.2g.success
- call.setup.mt.2g.attempts
- call.setup.mt.2g.fail.ratio
- call.setup.mt.2g.success.ratio
- call.setup.2g.fail.user
- call.setup.2g.fail.network
- call.setup.2g.fail
- call.setup.2g.success
- call.setup.2g.attempts
- call.setup.2g.fail.ratio
- call.setup.2g.success.ratio
- call.setup.2g.success.time.total
- call.setup.2g.success.time.attempts
- call.setup.2g.time.average
• call.setup.2g.time.max

2G dropped call
• call.dropped.mo.2g.fail.user
• call.dropped.mo.2g.fail.network
• call.dropped.mo.2g.fail
• call.dropped.mo.2g.success
• call.dropped.mo.2g.attempts
• call.dropped.mo.2g.fail.ratio
• call.dropped.mo.2g.success.ratio
• call.dropped.mt.2g.fail.user
• call.dropped.mt.2g.fail.network
• call.dropped.mt.2g.fail
• call.dropped.mt.2g.success
• call.dropped.mt.2g.attempts
• call.dropped.mt.2g.fail.ratio
• call.dropped.mt.2g.success.ratio
• call.dropped.2g.fail.user
• call.dropped.2g.fail.network
• call.dropped.2g.fail
• call.dropped.2g.success
• call.dropped.2g.attempts
• call.dropped.2g.fail.ratio
• call.dropped.2g.success.ratio

Bearer data attach
• bearer.data.2g.fail.user
• bearer.data.2g.fail.network
• bearer.data.2g.fail
• bearer.data.2g.success
• bearer.data.2g.attempts
• bearer.data.2g.fail.rate
• bearer.data.2g.fail.ratio
• bearer.data.3g.fail.user
• bearer.data.3g.fail.network
• bearer.data.3g.fail
Chapter 8: Using Metrics and Analyzing Events

- bearer.data.3g.success
- bearer.data.3g.attempts
- bearer.data.3g.fail.rate
- bearer.data.3g.success.rate
- bearer.data.combined.fail
- bearer.data.combined.fail.user
- bearer.data.combined.fail.network
- bearer.data.combined.success
- bearer.data.combined.attempts
- bearer.data.combined.fail.rate
- bearer.data.combined.fail.ratio
- bearer.data.combined.success.ratio

Bearer default/dedicated set-up time
- bearer.default.setup.success
- bearer.default.setup.time.total
- bearer.default.setup.time.mean
- bearer.dedicated.setup.success
- bearer.dedicated.setup.time.total
- bearer.dedicated.setup.time.mean

Bearer rejections
- bearer.setup.count
- bearer.setup.rejection.count
- bearer.setup.rejection.ratio

EPS attach
- eps.attach.fail.user
- eps.attach.fail.network
- eps.attach.fail
- eps.attach.success
- eps.attach.attempts
- eps.attach.success.rate
- eps.attach.fail.rate
- eps.attach.ratio

Gb
- pdp.context.2g.fail
- pdp.context.2g.fail.user
- pdp.context.2g.fail.network
- pdp.context.2g.success
- pdp.context.2g.attempts
- pdp.context.2g.fail.rate
- pdp.context.2g.success.rate

**GnGi session setup**
- session.setup.success
- session.setup.fail
- session.setup.attempts
- session.setup.fail.ratio
- session.setup.success.ratio
- session.setup.time.total
- session.setup.time.average

**GnGi network bandwidth**
- network.bandwidth.requested.total
- network.bandwidth.negotiated.total

**IuPS**
- pdp.context.3g.fail.user
- pdp.context.3g.fail.network
- pdp.context.3g.fail
- pdp.context.3g.success
- pdp.context.3g.attempts
- pdp.context.3g.fail.rate
- pdp.context.3g.success.rate

**Packets**
- packets.uplink
- packets.downlink
- packets.unknown
- packets.loss.ratio

**PDN connectivity**
- pdn.connect.fail.user
- pdn.connect.fail.network
- pdn.connect.success
Chapter 8: Using Metrics and Analyzing Events

- pdn.connect.fail
- pdn.connect.attempts
- pdn.connect.fail.rate
- pdn.connect.success.rate
- pdn.connect.ratio

**Radius data usage**
- radius.upload.octets.cumulative
- radius.download.octets.cumulative
- radius.upload.octets
- radius.upload.octets.attempts
- radius.download.octets
- radius.download.octets.attempts
- radius.upload.octets.rate
- radius.download.octets.rate
- radius.upload.octets.average
- radius.download.octets.average
- data.usage.session.time.total
- radius.upload.octets.session.rate
- radius.download.octets.session.rate

**Radius session summary**
- session.summary.device.4g
- session.summary.device.3g
- session.summary.device.2g
- session.summary.device.unknown
- session.summary.device.total
- session.summary.device.4g.rat.4g
- session.summary.device.3g.rat.3g
- session.summary.device.2g.rat.2g
- session.summary.device.4g.not.rat.4g
- session.summary.device.3g.not.rat.3g
- session.summary.rat.unknown
- session.summary.device.4g.not.rat.4g.cell.4g
- session.summary.device.4g.not.cell.4g
- session.summary.device.4g.not.rat.4g.cell.unknown
Chapter 8: Using Metrics and Analyzing Events

- session.summary.device.3g.not.rat.3g.cell.3g
- session.summary.device.3g.not.rat.3g.cell.3g
- session.summary.device.3g.not.rat.3g.cell.unknown
- session.summary.device.rate
- session.summary.device.4g.rat.ratio
- session.summary.device.3g.rat.ratio
- session.summary.device.4g.not.rat.4g.ratio
- session.summary.device.3g.not.rat.3g.ratio
- session.summary.device.4g.rat.4g.ratio
- session.summary.device.3g.rat.3g.ratio

**Radius session terminate**
- radius.terminate.fail.user
- radius.terminate.fail.network
- radius.terminate.success
- radius.terminate.fail
- radius.terminate.attempts
- radius.terminate.fail.rate
- radius.terminate.fail.ratio
- radius.terminate.success.ratio

**Retransmission**
- retransmission.packets.uplink
- retransmission.packets.downlink
- retransmission.time
- retransmission.packets
- retransmission.rate

**S1-MME default/dedicated bearer activate**
- eps.defaultbearer.activate.fail
- eps.defaultbearer.activate.success
- eps.defaultbearer.activate.attempts
- eps.defaultbearer.activate.fail.rate
- eps.defaultbearer.activate.success.rate
- eps.defaultbearer.activate.ratio
- eps.dedicatedbearer.activate.fail
- eps.dedicatedbearer.activate.success
Chapter 8: Using Metrics and Analyzing Events

- eps.dedicated.bearer.activate.attempts
- eps.dedicated.bearer.activate.fail.rate
- eps.dedicated.bearer.activate.success.rate
- eps.dedicated.bearer.activate.ratio
- eps.bearer.activate.fail
- eps.bearer.activate.success
- eps.bearer.activate.attempts
- eps.bearer.activate.fail.rate
- eps.bearer.activate.success.rate
- eps.bearer.activate.ratio

**S1-MME default/dedicated bearer detach**
- eps.detach.fail
- eps.detach.success
- eps.detach.attempts
- eps.detach.fail.rate
- eps.detach.success.rate
- eps.detach.ratio

**Session drop**
- session.drop.fail
- session.drop.status.fail
- session.drop.success
- session.drop.attempts
- session.drop.status.attempts
- session.drop.ratio
- session.drop.status.ratio

**SMS 2G**
- SMS 2G submission:
  - sms.mo.2g.fail
  - sms.mo.2g.fail.user
  - sms.mo.2g.fail.network
  - sms.mo.2g.success
  - sms.mo.2g.attempts
  - sms.mo.2g.fail.rate
  - sms.mo.2g.success.rate
SMS 2G delivery:
- sms.mt.2g.fail
- sms.mt.2g.fail.user
- sms.mt.2g.fail.network
- sms.mt.2g.success
- sms.mt.2g.attempts
- sms.mt.2g.fail.rate
- sms.mt.2g.success.rate

SMS 3G
- SMS 3G submission:
  - sms.mo.3g.fail
  - sms.mo.3g.fail.user
  - sms.mo.3g.fail.network
  - sms.mo.3g.success
  - sms.mo.3g.attempts
  - sms.mo.3g.fail.rate
  - sms.mo.3g.success.rate
- SMS 3G delivery:
  - sms.mt.3g.fail
  - sms.mt.3g.fail.user
  - sms.mt.3g.fail.network
  - sms.mt.3g.success
  - sms.mt.3g.attempts
  - sms.mt.3g.fail.rate
  - sms.mt.3g.success.rate

SMS combined
- SMS combined submission:
  - sms.mo.combined.fail
  - sms.mo.combined.fail.user
  - sms.mo.combined.fail.network
  - sms.mo.combined.success
  - sms.mo.combined.attempts
  - sms.mo.combined.fail.rate
  - sms.mo.combined.success.rate
• SMS combined delivery:
  ▪ sms.mt.combined.fail
  ▪ sms.mt.combined.fail.user
  ▪ sms.mt.combined.fail.network
  ▪ sms.mt.combined.success
  ▪ sms.mt.combined.attempts
  ▪ sms.mt.combined.fail.rate
  ▪ sms.mt.combined.success.rate

• SMS combined submission and delivery:
  ▪ sms.combined.attempts
  ▪ sms.combined.fail
  ▪ sms.combined.fail.rate
  ▪ sms.combined.success.rate

Tracking area update
• tau.periodic.fail.user
• tau.periodic.fail.network
• tau.periodic.success
• tau.periodic.fail
• tau.periodic.attempts
• tau.periodic.fail.rate
• tau.periodic.success.rate
• tau.periodic.ratio
• tau.nonperiodic.fail.user
• tau.nonperiodic.fail.network
• tau.nonperiodic.success
• tau.nonperiodic.fail
• tau.nonperiodic.attempts
• tau.nonperiodic.fail.rate
• tau.nonperiodic.success.rate
• tau.nonperiodic.ratio
• tau.combined.fail.user
• tau.combined.fail.network
• tau.combined.success
• tau.combined.fail
Chapter 8: Using Metrics and Analyzing Events

- tau.combined.attempts
- tau.combined.fail.rate
- tau.combined.success.rate
- tau.combined.ratio
- tau.combined.attach.fail.user
- tau.combined.attach.fail.network
- tau.combined.attach.success
- tau.combined.attach.fail
- tau.combined.attach.attempts
- tau.combined.attach.fail.rate
- tau.combined.attach.success.rate
- tau.combined.attach.ratio
- tau.fail.user
- tau.fail.network
- tau.success
- tau.fail
- tau.attempts
- tau.fail.rate
- tau.success.rate
- tau.ratio

Upload/download speeds

- upload.download.negotiated.bandwidth.total
- upload.download.requested.bandwidth.total
- upload.download.bandwidth.gap

XDATA

- pdp.context.combined.fail
- pdp.context.combined.fail.user
- pdp.context.combined.fail.network
- pdp.context.combined.success
- pdp.context.combined.attempts
- pdp.context.combined.fail.rate
- pdp.context.combined.success.rate

See Available metrics for the complete list of available metrics components for inclusion in metrics streams.
Chapter 8: Using Metrics and Analyzing Events

Common options

\texttt{rti.distgrid.metrics.<stream>.enabled=<true/false>}

Turns a stream on or off. Only the \texttt{celltower} stream is enabled by default.

\texttt{rti.distgrid.metrics.<stream>.sizingHint=<Integer>}

The approximate number of planned metrics entities, given the reference data and key expression. Round up.

\textit{sizingHint} provides RTI metrics with information on the expected unique grouping keys. This enables the system to allocate internal processing resources up front to reduce in-process resource allocation overhead. This property does not limit the number of actual records and the system can accommodate more records dynamically.

Default value: \texttt{75000}.

\texttt{rti.distgrid.metrics.<stream>.include.patterns=<Pattern>}
\texttt{rti.distgrid.metrics.<stream>.exclude.patterns=<Pattern>}

The metrics counters and expressions to include in the stream. This option supports Ant pattern matcher syntax:

- \texttt{?} matches one character
- \texttt{*} matches zero or more characters
- \texttt{**} matches zero or more packages or sub-packages

For example:

- \texttt{sms.mo.2g.*.rate} includes \texttt{sms.mo.2g.fail.rate} and \texttt{sms.mo.2g.success.rate}.
- \texttt{sms.**} includes all metrics in the \texttt{sms} package and sub-packages.

\textbf{Important:} Enable only the metrics components that you want to track to avoid impacting system performance negatively.

\texttt{rti.distgrid.metrics.<stream>.export.format=<MetricsComponents>}

A comma-separated list of the metrics to write to the CSV file or AMQP exchange. List the metrics in the desired order in the output.

See \textit{Available metrics} for the complete list of available metrics components for inclusion in metrics streams.

The following expressions are supported:

- \texttt{ref.xxx}: A field from the ref data object attached to the metrics. See ref data objects.
- \texttt{a.b.c}: A specific metric
  - \texttt{*}: All enabled metrics

For example:

\texttt{a.b.c, e.f.g, ref.name, *sms.mo.2g.fail,sms.mo.2g.fail.user,sms.mo.2g.fail.network,sms.mo.2g.success,sms}
Default value: \*.

\[\text{Note: } \text{Row breaks in the comma-separated list are not permitted.}\]

The \* wildcard can be used anywhere in the list. For example, a,b,\*,z will produce a,b,c,x,f,e,g,z. The ordering of the \*:ed columns is undefined.

\[rti.distgrid.metrics.<stream>.eventTime=EVENT\_TIME\]

A time bucket is the time period associated to a metric. The system retains and calculates metrics within a specific time bucket. This property determines the reference point used to determine the correct time bucket for an event. The system standardizes on this reference point when adding the metrics for an event.

Options include:

- \text{EVENT\_START\_TIME}: Data record in the event.
- \text{EVENT\_TIME}: Data record in the event.
- \text{INGEST\_TIME}: Computer clock when the event is processed in the ingest grid.
- \text{SYSTEM\_TIME}: Computer clock when the event is processed in the distribution grid. Only intended for testing, not for production use.

Default value: \text{EVENT\_TIME}.

\[rti.distgrid.metrics.<stream>.slide.period=<TimePeriod>\]

The frequency (in milliseconds) with which a slide occurs. A slide progresses the window, performs metrics calculations on the updated window, and then exports the updated metrics. Decrease this value to 30000 (a half minute) when testing the metrics functionality.

Default value: 900000.

\[rti.distgrid.metrics.<stream>.window.length=<TimePeriod>\]

The length of the time window. A value of 60000 would create a window of the metrics in the past minute. While testing the metrics functionality, try reducing this period to 30000 milliseconds (a half minute).

Default value: 900000 (15 minutes).

\[rti.distgrid.metrics.pr.redundancy=<Number>\]

Determines whether to create a redundant copy of each metric in Pivotal GemFire. Options:

- 0: No redundancy
- 1: A redundant copy of each metric is stored.

Default value: 0.

\[rti.distgrid.metrics.file.compress=<true/false>\]

Turn metrics output compression on or off. By default, the metrics output is compressed. Options:
Chapter 8: Using Metrics and Analyzing Events

- True: Compress
- False: Do not compress.

Default value: true.

\texttt{rti.distgrid.metrics.amqp.output=<true / false>}

When true, will output metrics to AMQP. When false, RTI will not output metrics data to AMQP.

Options:

- True: Send metrics output to AMQP.
- False: Do not send metrics output to AMQP.

Default value: true.

\texttt{rti.distgrid.metrics.file.output}

When true, metrics output is sent to a file. When false, RTI will not output metrics data to file.

Options:

- True: Send metrics output to a file.
- False: Do not send metrics output to a file.

Default value: true.

The following properties are deprecated and no longer used:

- \texttt{rti.distgrid.metrics.<stream>.shuffle.period}
- \texttt{rti.distgrid.metrics.<stream>.save.period}
- \texttt{rti.distgrid.metrics.<stream>.export.period}
- \texttt{rti.distgrid.metrics.<stream>.timebucket.size}
- \texttt{rti.distgrid.metrics.<stream>.timebuckets.history}
- \texttt{rti.distgrid.metrics.<stream>.timebuckets.future}

**Cell tower stream options**

\texttt{rti.distgrid.metrics.celltower.groupBy.expression=<Value>}

Sets the granularity of the celltower stream:

- cellKey: A cellKey is a string composed of the lac and the celltower id (that is, \texttt{<lac>-<celltower_id>}). Each celltower is tracked individually. Enables the use of \texttt{ref.xxx} expressions in the exported CSV file.
- lac: The celltower metric streams are tracked for each LAC/TAC (Type Approval Code) area.

\begin{itemize}
\item \textbf{Note:} Celltower reference data is not available in CSV output with the \texttt{lac} option.
\end{itemize}
**rti.distgrid.metrics.celltower.export.format=<Reference Data>**

Available reference data: See CellTower in Reference data types. For example:

`ref.cellName,ref.sitePostCode,*`

**Device stream options**

**rti.distgrid.metrics.device.groupBy.expression=<SpEL Expression>**

SpEL expression to pick the device key:

**Note:** For these expressions to work, the proper MobileEquipment ref data needs to be loaded into RTI. If no ref data exists for a message, its grouping key will be "NA". See the MobileEquipment reference data type.

- `mobileEquipment.tac`: Fine-grained per mobile device subtype. Enables the use of ref.xxx expressions in the exported CSV file or AMQP message.
- `mobileEquipment.model`: Mobile equipment model used, such as iPhone.

**Note:** Reference data is not available in CSV output for the mobileEquipment.model option.

- `mobileEquipment.manufacturer`: Manufacturer of mobile equipment used, such as Apple.

**Note:** Reference data is not available in CSV output for the mobileEquipment.manufacturer option.

Combinations are supported: For example:

Combining expressions creates a compound key:

`mobileEquipment.manufacturer + '-' + mobileEquipment.model`

**rti.distgrid.metrics.device.export.format=ref.manufacturer,ref.model,***

Available reference data: See MobileEquipment in Reference data types.

**Subscriber groups stream options**

None.

**Mobile networks stream options**

**rti.distgrid.metrics.mobilenetwork.groupBy.expression=<SpEL Expression>**

SpEL expression to pick the network key:

- `mccmnc`
- `mcc`
- `mnc`

**rti.distgrid.metrics.mobilenetwork.export.format=ref.manufacturer,ref.model,***

Available reference data: See MobileEquipment in Reference data types.

**Consuming metrics**

**Consuming a metrics stream as files**

Metrics output files are organized by domain type:
Chapter 8: Using Metrics and Analyzing Events

- `<metric-stream>/`
  - `<time-bucket>/`
    - `<metrics-stream>.<start-time-bucket>.<end-time-bucket>.<export-time>.<node>.csv.gz`

For example:

- `celltower/`
- `20140401T143015Z/`
- `celltower.20140401T143015Z.20140401T143030Z.20140401T143145Z.distgrid1.csv.gz`

Where:

- `<start-time-bucket>` is the start time of the time bucket.
- `<end-time-bucket>` is the end time of the time bucket.
- `<export-time>` is the time this metrics file was exported.
- `<node>` is the RTI node that exported the file. This variable makes the files unique if collated.

The export time typically occurs after the time bucket is completed. Each time bucket will be output to one or more separate files.

**Metrics file format**

Output files are in CSV format. Included columns are determined by RTI-T configuration per metrics flow. The files can be optionally compressed.

**Header**

ID,<metric-name>,<metric-name>

For example:

ID,upload-bandwidth,download-bytes

**Payloads**

<k>, <metric>, <metric>

For example:

123-76432, 10485760, 52428800

**Consuming a metrics stream in an AMQP exchange**

AMQP (Advanced Message Queuing Protocol) is an open standard application layer protocol for message-oriented middleware. RTI can push metrics data about enabled streams to an AMQP exchange named `decoder.metrics`. Each message is published to the exchange with the stream name as the routing key.
Create a RabbitMQ queue and bind this queue to the `decoder.metrics` exchange with the stream name as the routing key. Different enumerations of the attributes in enabled streams share the same stream name. All of these enumerations are included in the exchange with a single routing key. RTI currently supports streams for cell towers and devices.

**Routing keys:**
- Cell tower: `cellTower`
- Devices: `device`

See [RTISH commands](#) for the RTISH commands for creating or deleting AMQP exchanges and queues.

Metrics messages sent out via AMQP contain the following headers:
- `x-rti-metrics-timeBucket-exportDate`: The datetime when the export was performed (in ISO 8601 format).
- `x-rti-metrics-timeBucket-startDate`: The datetime of the beginning of the time bucket (in ISO 8601 format).
- `x-rti-metrics-timeBucket-endDate`: The datetime of the end of the time bucket (in ISO 8601 format).
- `x-rti-metrics-streamName`: The name of the stream. For example, `celltower`.

### Metrics statistics

GemFire statistics provide insight into data distribution and JVM performance. The set of available statistics encompass real-time metric computation to provide a holistic view of the system at large.

Enable GemFire statistics by running the RTISH `set property` command to configure the following properties in the `config.properties` file. See [Configuration properties](#) for instructions on running the command.

- `rti.gemfire.enable-time-statistics=true`
- `rti.gemfire.statistic-sampling-enabled=true`
- `rti.gemfire.statistic-archive-file=${rti.logs}/rti-distgrid-gemfire-stats.gfs`
- `rti.gemfire.statistic-sample-rate=5000`

See the GemFire documentation for information about these properties: [http://gemfire.docs.pivotal.io/index.html](http://gemfire.docs.pivotal.io/index.html)

Statistics for metric processing are performed at three levels of metric computation:
- Export
- Stream
- Engine

These statistics can be used to validate and debug processing or understand how the system is performing with respect to load while overlaying additional platform statistics.
You can enable both course- and fine-grained runtime statistical information for configured metric streams. Statistical counters are not enabled by default.

**Coarse-grained statistics**

Stream-level statistics are updated during the metric export process. Enable statistics gathering for a specific metrics stream by running the following command in RTISH:

```bash
rti set properties
--name rti.distgrid.metrics.<streamName>.statistics.enabled --value true
--instances distgrid@
```

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Statistical counters</th>
</tr>
</thead>
<tbody>
<tr>
<td>MetricsExporterStats</td>
<td>&lt;stream.name&gt;&lt;exporter-type&gt;</td>
<td>skippedBuckets, Total number of exported metric buckets exported, timeBucketsSent Number of skipped buckets per export</td>
</tr>
<tr>
<td>Stream-level statistics</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Fine-grained statistics**

A metrics stream can result from a formula applied to a collection of independent components. These components are monitored to ensure that they are updated as events occur. Statistics are generated for the various processing stages of a stream.

Enable statistics gathering for a specific metrics stream by running the following command in RTISH:

```bash
rti set properties --name
rti.distgrid.metrics.&lt;streamName&gt;.components.statistics.enabled --value true --instances distgrid@
```
Table 15. Fine-grained metrics statistics

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Statistical counters</th>
</tr>
</thead>
<tbody>
<tr>
<td>MetricComponentStatistics</td>
<td>MetricComponentStats-</td>
<td><strong>Invoked</strong>: Number of times the component was invoked by the update function.</td>
</tr>
<tr>
<td></td>
<td>mobilenetwork-tau.ratio</td>
<td><strong>Hits</strong>: Number of times the component was updated successfully.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Skipped</strong>: Number of times an update failed. If the number exceeds zero, then a</td>
</tr>
<tr>
<td></td>
<td></td>
<td>system error occurred. Refer to system log file, and contact Customer Support.</td>
</tr>
<tr>
<td>MetricStreamStats (Map)</td>
<td>&lt;streamName&gt;</td>
<td><strong>mapEngineOutsideTimeWindowEvent</strong>: Total number of events with timestamps outside</td>
</tr>
<tr>
<td></td>
<td></td>
<td>the current time window</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>mapEngineMetricsObjects</strong>: Number of metrics components in use</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>mapEngineTimeBucketsUsed</strong>: Number of time buckets in use</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>mapEngineTimeBucketsPooled</strong>: Number of time buckets pooled</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>mapEngineTimeBucketsDiscarded</strong>: Number of time buckets discarded and not pooled</td>
</tr>
<tr>
<td>MetricStreamStats (Reduce)</td>
<td>&lt;streamName&gt;</td>
<td><strong>reduceEngineMetricsObjects</strong>: Number of metric components in use</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>reduceEngineTimeBucketsUsed</strong>: Number of time buckets in use</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>reduceEngineTimeBucketsPooled</strong>: Number of time buckets pooled</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>reduceEngineTimeBucketsDiscarded</strong>: Number of time buckets discarded and not pooled</td>
</tr>
</tbody>
</table>

**Metrics engine statistics**

Statistics are published to the standard RTI log file on a defined cyclical period to provide insight into metric engine performance with respect to system load. Specify the frequency (in minutes, using the m suffix) with which the metrics engine should write the log records. Use RTISH to set the property on the distribution grid nodes.

See **RTI log files** for information about RTI log files.

Enable statistics gathering for the metrics engine by running the following command in RTISH:

```bash
rti set properties --name rti.distgrid.statistics.logPeriod --value <TimePeriod>
```

The time period is calculated in seconds. For example, a value of **300** is equivalent to 5 minutes.

The following example provides statistical information on how the `mobilenetwork.reduce.saveFilter` process is performing:
13:04:18.279 INFO [metrics-logger-reporter-thread-1] metrics - type=TIMER,
name=mobilenetwork.reduce.saveFilter, count=37, min=0.008,
max=0.088, mean=0.029972972972972974, stddev=0.011837995528726058,
median=0.03, p75=0.032,
p95=0.04300000000000007, p98=0.088, p99=0.088, p999=0.088,
mean_rate=0.1226194799158389, m1=0.10793225621931721,
m5=0.21943373017673273,
m15=0.3194621117944544, rate_unit=events/second,
duration_unit=millisecons

where:

- **count**: The number of invocations.
- **min**: The minimum invocation time. The time unit is specified by `duration_unit`.
- **max**: The maximum invocation time. The time unit is specified by `duration_unit`.
- **mean**: The mean deviation for the invocation time.
- **stddev**: The standard deviation for the invocation time.
- **p75, p95, and so on**: The percentile of invocation times. This information is useful to watch for alerting.

The following timers are available:

- **<stream>.map.shuffleExtract**: The length of time to extract and reset one batch of metrics. If this value exceeds 100 milliseconds, then the ingestion of events into the metrics engine may suffer. Consider using coarser-grained metrics.

- **<stream>.map.shufflePrep**: The length of time the shuffle process waits for the `shuffleExtract`. If this value runs into the 10s of seconds, then the reactor is backlogged, and the queue size must be tuned.

- **<stream>.map.shuffle**: The length of time to send one batch of metrics to the correct grid nodes for accumulations. This timer runs in a different thread from `metrics ingest` but does not require more than a few seconds. If the timer takes longer, then investigate the reduction process using virtual file system (VFS) statistics.

- **<stream>.map.timeBucketRecycle**: The length of time to reap and recycle time buckets internally. This timer runs in under 100 milliseconds.

- **<stream>.reduce.reduce**: The length of time to complete the entire process of calculating derived metrics. This timer requires between 0.5 second and a few seconds. If the timer requires more than 5 seconds, then consider reducing the number of derived metrics that you have enabled, or increase the shuffle period setting to a value 2 to 3 times higher than the reduction time.

- **<stream>.reduce.timeBucketRecycle**: The length of time to reap and recycle time buckets internally. This timer runs in under 100 milliseconds.

- **<stream>.reduce.export**: The length of time for the export process to complete. The export process runs in a different thread from the reduction process. The length
of time required depends on how coarse-grained the metrics are and the speed of downstream components. This timer does not require more than 10s of seconds.

- `<stream>.reduce.saveFilter`: The length of time to filter out the "dirty" (changed) metrics component for persistence into Pivotal GemFire. This timer runs in under 100 milliseconds.

- `<stream>.reduce.save`: The length of time to persist dirty metrics components to GemFire for high availability (HA). The length depends highly on the number of redundant copies and on how busy GemFire is. This timer should run in under 5 seconds; if not, investigate GemFire performance using VFS.

Creating user-defined metrics streams

You can create user-defined metrics streams. This is described in the *EMC Real-Time Intelligence (RTI) Developers’ Guide*.

Summary of RTISH Commands to control user defined metrics streams

Metric streams are managed using the RTISH command line interface. This allows for end users to manage and deploy new streams in to the system. This section lists the available commands and usages. Detailed descriptions for each command are available in the RTISH commands section.

Available RTISH commands include:

- deploy -- rti metrics deploy
- undeploy -- rti metrics undeploy
- enable -- rti metrics enable
- disable -- rti metrics disable
- list -- rti metrics list
- describe -- rti metrics describe
- reload -- rti metrics reload

In addition to RTISH commands, metrics streams can be managed remotely through the REST API application. To manage metrics streams, the REST API application must be connected to the distribution grid. For details about creating and starting the REST API application, refer to Setting up and running the REST client section in this document and Creating the rest-api application section in the *EMC Real-Time Intelligence (RTI) Streaming Guide*.

**Deploy a metrics stream**

Metrics stream deployment requires a valid metrics JSON definition file. The following command deploys a metrics stream across the distributed cluster. At system restart, all deployed and enabled metrics streams start and are ready for processing.

**REST URL:**

<hostAddress>:<port>/rti-t/kpi/deploy
**CURL command:**

curl -F stream=@/path/to/stream/your_stream.json <REST URL>

**Example:**

curl -F stream=@/Users/affirmed/affirmed_metrics_stream.json localhost:9001/rti-t/kpi/deploy

**Response:**

affirmed_metrics_stream successfully started.

**List a metrics stream**
The following command lists all metrics streams.

**REST URL:**

<hostAddress>:<port>/rti-t/kpi/list

**CURL command:**

curl <REST URL>

**Example:**

curl localhost:9001/rti-t/kpi/list

**Response:**

Note: Depends on the deployed metrics streams, it may have more streams in the list.

<table>
<thead>
<tr>
<th>Name</th>
<th>Active</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>affirmed-metrics-stream</td>
<td>true</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>
Enable a metrics stream
The following command enables a metrics stream.

**REST URL:**

```
<hostAddress>:<port>/rti-t/kpi/enable?name=<metrics-stream-name>
```

**CURL command:**

```
curl -X POST <REST URL>
```

**Example:**

```
curl -X POST localhost:9001/rti-t/kpi/enable?name=affirmed-metrics-stream
```

**Response:**

affirmed-metrics-stream successfully enabled.

Disable a metrics stream
The following command disables a metrics streams.

**REST URL:**

```
<hostAddress>:<port>/rti-t/kpi/disable?name=<metrics-stream-name>
```

**CURL command:**

```
curl -X POST <REST URL>
```

**Example:**

```
curl -X POST localhost:9001/rti-t/kpi/disable?name=affirmed-metrics-stream
```

**Response:**

affirmed-metrics-stream successfully disabled.

Describe a metrics stream
The following command describes a metrics streams.

**REST URL:**

```
<hostAddress>:<port>/rti-t/kpi/describe?name=<metrics-stream-name>
```

**CURL command:**

```
curl <REST URL>
```

**Example:**

```
curl localhost:9001/rti-t/kpi/describe?name=affirmed-metrics-stream
```
**Response:**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>affirmed-metrics-stream</td>
</tr>
<tr>
<td>enabled</td>
<td>false</td>
</tr>
</tbody>
</table>

**Undeploy a metrics stream**

The following command undeploys a metrics stream.

**REST URL:**

```
<hostAddress>:<port>/rti-t/kpi/undeploy?name=<metrics-stream-name>
```

**CURL command:**

```
curl -X POST <REST URL>
```

**Example:**

```
curl -X POST localhost:9001/rti-t/kpi/undeploy?name=affirmed-metrics-stream
```

**Response:**

affirmed-metrics-stream successfully undeployed.

---

**System startup notes**

This section explains how the metric subsystem starts up both on initial installation and on system restart situations.

**Post installation startup**

When the system has been installed and activated, default metric stream definitions are deployed into the system. You have the choice of deactivating or enabling any of these definitions using the metrics-specific RTISH commands.

**System restart**

When the system restarts, previous system deployments are reinstated. User-defined metric streams are also redeployed in their previous configured state.

**User-defined metrics streams troubleshooting**

**Error codes**

This section provides the Metric System error codes. Capture these conditions by monitoring the cluster log files.

**E1000**

Invalid 'where' expression. Review the syntax of the expression and resubmit using RTISH.
**KPI store function**

KPI store

RTI calculates KPI metrics through the KPI/metrics engine. After KPI results are calculated, KPI store saves them so that the historical metrics results can be referenced as needed.

The KPI results are calculated based on “group by” values. Some metrics could be calculated based on Cell Tower (group by CellTowerID). Meanwhile, other metrics could be calculated based on subscriber ID (group by IMSI). The group by values of CellTowerID or IMSI are different in scope: there could be millions of subscribers, while there would be fewer cell towers. To optimize KPI data store performance, considering the differences in group-by values, we maintain two KPI store regions in each RTI distribution grid instance. One KPI store region is a replicated region to save the KPI results with smaller group-by values. The other KPI store region is a partitioned region used to save the KPI results with large group-by values.

KPI store checks the KPI result every minute and expires the KPI results that have a timestamp before (currentTime – TTL). For example, if TTL is 43,200 minutes (30 days), KPI results received more than 30 days ago will be removed.

For further information on how to use the KPI Store feature, refer to the *EMC Real-Time Intelligence (RTI) Streaming Guide* and *EMC Real-Time Intelligence (RTI) Developers Guide*. 
This chapter presents the following topics:

- Exporting data ................................................................. 139
- Using AMQP exchange ..................................................... 139
- Using a CSV file ................................................................. 142
- Using a JDBC data source .................................................... 143
- Using HDFS transport to export events into Hadoop ................. 144
- InfluxDB transport ............................................................... 155
Exporting data

This chapter describes the different ways to export data from RTI using:

- AMQP exchange
- A CSV file
- A JDBC data source
- HDFS Transport into Hadoop

Using AMQP exchange

AMQP (Advanced Message Queuing Protocol) is an open standard application layer protocol for message oriented middleware. RTI pushes location-based event stream data to a configured AMQP exchange.

Each location-based event stream data is published to the AMQP exchange with a unique routing key. See Provisioning Manager and RabbitMQ properties for the list of configuration properties.

To consume LORS or LANDS event stream data, the RabbitMQ queue should be bound to the RabbitMQ exchange. Different enumerations of the LANDS or LORS event stream data are published from the distribution grid to the RabbitMQ exchange DECODER.STREAM with one of the following routing keys:

- DECODER.STREAM.ALL.LORS
- DECODER.STREAM.ALL.LANDS

The Provisioning Manager configures the RabbitMQ exchange with type TOPIC to support Pub/Sub (publish–subscribe) routing. The routing of event stream data from the RabbitMQ exchange to different queues is based on the multiple routing key values set in the BCC property of the message header, enriched as part of the Grid2 message flow. The RabbitMQ exchange examines the BCC message property in the header field and routes messages to all queues matching the routing key. The event stream data type in the BCC header value is an array of longstr which includes the values of routing keys prefixed with the following value:

<OrgUnit>.<SubscriptionID>
The following diagram provides an example of message routing in RabbitMQ when events are published from the distribution grid:

![Diagram](image)

**Figure 15. Catch All Message Routing**

Following is a table of Grid2 RabbitMQ configuration parameters:

**Table 16. Distribution grid RabbitMQ configuration parameters**

<table>
<thead>
<tr>
<th>Description</th>
<th>Distribution grid property</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMQP exchange name</td>
<td>rti.distgrid.lors.amqp.exchange</td>
<td>DECODER.STREAM</td>
</tr>
<tr>
<td></td>
<td>rti.distgrid.lands.amqp.exchange</td>
<td></td>
</tr>
<tr>
<td>AMQP routing key</td>
<td>rti.distgrid.lors.amqp.routingKey</td>
<td>DECODER.STREAM.ALL.LORS</td>
</tr>
<tr>
<td></td>
<td>rti.distgrid.lands.amqp.routingKey</td>
<td></td>
</tr>
<tr>
<td>Content type of message payload</td>
<td>rti.distgrid.lors.amqp.contentType</td>
<td>text/csv</td>
</tr>
<tr>
<td></td>
<td>rti.distgrid.lands.amqp.contentType</td>
<td></td>
</tr>
</tbody>
</table>

Following is a table of Grid2 RabbitMQ durability parameters:
Chapter 9: Exporting Data

Table 17. Distribution grid RabbitMQ durability parameters

<table>
<thead>
<tr>
<th>Distribution grid property</th>
<th>Default value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rti.distgrid.rabbit.catchAllExchange.durable</td>
<td>TRUE</td>
<td>Controls whether the catchAll RMQ Exchange is durable.</td>
</tr>
<tr>
<td>rti.distgrid.rabbit.catchAllQueue.durable</td>
<td>TRUE</td>
<td>Controls whether the cacheAll RMQ Queue is durable.</td>
</tr>
<tr>
<td>rti.distgrid.ess.rabbit.queue.durable</td>
<td>TRUE</td>
<td>Controls whether the per-ESS RMQ queues are durable or not.</td>
</tr>
</tbody>
</table>

Run the following RTISH command to update a property value:

```
rti set property --instances <name@host> --name <propertyName> --value <propertyValue>
```

For example:

```
rti set property --instances distgrid-1@localhost --name rti.distgrid.lors.amqp.exchange --value DECODER.STREAM
```

The routing key of the RTI AMQP message is `DECODER.STREAM.ALL`.

The following table has a list of RTI AMQP message headers:

Table 18. RTI AMQP message headers

<table>
<thead>
<tr>
<th>Header</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>decoder.messageHash</td>
<td>Hash value of the message payload</td>
</tr>
<tr>
<td>decoder.eventCount</td>
<td>Number of event stream payload in the message</td>
</tr>
<tr>
<td>decoder.payloadFormat</td>
<td>Output format of the message</td>
</tr>
<tr>
<td>rti.version</td>
<td>RTI version</td>
</tr>
<tr>
<td>clusterid</td>
<td>Cluster ID of originating message. Set to distgrid.</td>
</tr>
<tr>
<td>contentEncoding</td>
<td>Content encoding. Set to UTF-8.</td>
</tr>
<tr>
<td>application id</td>
<td>Set to distgrid.</td>
</tr>
<tr>
<td>deliveryMode</td>
<td>Set as PERSISTENT</td>
</tr>
<tr>
<td>BCC</td>
<td>Array of routing keys</td>
</tr>
</tbody>
</table>

For debugging purposes, the `rti show output` RTISH command produces information about the current message from the distribution grid. See `rti show output` for more information.
Using a CSV file

When enabled, events are written to a CSV file. The file name is comprised of the following parts:

- Type name:
  - LORS
  - LANDS
- Instance name (Default: distgrid)
- Formatted date (Default: `yyyyMMdd'T'HHmmssSSS`). See the "Data format" properties in Table 19.
- .csv extension

Example: LORS-distgrid-one@localhost-20141015T082813887.csv

Following is the list of Grid2 file output configuration parameters:

<table>
<thead>
<tr>
<th>Description</th>
<th>Distribution grid property</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable file output</td>
<td>rti.distgrid.lors.file.enabled</td>
<td>false</td>
</tr>
<tr>
<td></td>
<td>rti.distgrid.lands.file.enabled</td>
<td></td>
</tr>
<tr>
<td>Date format</td>
<td>rti.distgrid.lors.file.dateFormat</td>
<td><code>yyyyMMdd'T'HHmmssSSS</code></td>
</tr>
<tr>
<td></td>
<td>rti.distgrid.lands.file.dateFormat</td>
<td></td>
</tr>
<tr>
<td>File path</td>
<td>rti.distgrid.lors.file.path</td>
<td><code>${rti.data}/lors</code></td>
</tr>
<tr>
<td></td>
<td>rti.distgrid.lands.file.path</td>
<td><code>${rti.data}/lands</code></td>
</tr>
<tr>
<td>Batch size. The number of events in a batch. The value must be greater than 0.</td>
<td>rti.distgrid.lors.file.batchSize</td>
<td>8192</td>
</tr>
<tr>
<td></td>
<td>rti.distgrid.lands.file.batchSize</td>
<td></td>
</tr>
<tr>
<td>Batch time out. The length of time (in milliseconds) before the full batch size is reached and the batch of events are written to file. The value must be greater than 1.</td>
<td>rti.distgrid.lors.file.timeout</td>
<td>4000</td>
</tr>
<tr>
<td></td>
<td>rti.distgrid.lors.file.timeout</td>
<td></td>
</tr>
<tr>
<td>File compression</td>
<td>rti.distgrid.lors.file.compress</td>
<td>false</td>
</tr>
<tr>
<td></td>
<td>rti.distgrid.lands.file.compress</td>
<td></td>
</tr>
<tr>
<td>Delay (in milliseconds) until the compression occurs. A value of at least 1000 overrides any value set lower.</td>
<td>rti.distgrid.lors.file.compressionDelay</td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td>rti.distgrid.lands.file.compressionDelay</td>
<td></td>
</tr>
</tbody>
</table>
Run the following RTISH command to update a property value:

```bash
rti set property --instances <name@host> --name <propertyName> --value <propertyValue>
```

For example:

```bash
rti set property --instances distgrid-1@localhost --name rti.distgrid.lors.file.batchSize --value 8000
```

### Using a JDBC data source

JDBC output is available to LORS processing only. When enabled, events are written to a JDBC data source.

The properties in the following table, which shows Grid2 file configuration parameters, are configurable.

<table>
<thead>
<tr>
<th>Description</th>
<th>Distribution grid property</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>INSERT statement</td>
<td>rti.distgrid.lors.jdbc.insertStatement</td>
<td>INSERT INTO RTI.RTI_LANDING (payload, feed) VALUES (?, ?)</td>
</tr>
<tr>
<td>Batch size. The number of events in a batch. The value must be greater than 0.</td>
<td>rti.distgrid.lors.jdbc.batchSize</td>
<td>2048</td>
</tr>
<tr>
<td>Batch time out. The length of time (in milliseconds) before the full batch size is reached and the batch of events are written to JDBC. The value must be greater than 1.</td>
<td>rti.distgrid.lors.jdbc.timeout</td>
<td>4000</td>
</tr>
</tbody>
</table>

The data source, connection, driver and transactional attributes are also fully configurable. For the complete list of JDBC properties, see Configuration under the Whitelists and Blacklists section in the RTI Concepts Guide.

Run the following RTISH command to update a property value:

```bash
rti set property --instances <name@host> --name <propertyName> --value <propertyValue>
```

For example:

```bash
rti set property --instances distgrid-1@localhost --name rti.distgrid.lors.jdbc.batchSize --value 2048
```
Using HDFS transport to export events into Hadoop

RTI HDFS Transport Application is a sub-system for exporting the events produced by RTI into Hadoop. Once configured as part of an RTI deployment, this application automatically exports enabled event streams into Hadoop without interfering with the existing process flow in RTI. This data can be then used for downstream map-reduce applications or other kinds of Hadoop workloads. More information about the various processes involved in setting HDFS Transport can be found in the following sections.

Notes:

HDFS Transport application is integrated through RTISH and has the same lifecycle of any other RTISH-managed RTI application.

HDFS Transport does not impact the critical path flow of RTI event stream processing, as it receives data from the downstream AMQP transports.

The RTI HDFS Transport is currently certified against the following versions of Hadoop:

- Apache Hadoop 2.2.0
- Apache Hadoop 2.4.0
- Apache Hadoop 2.5.2

There is also experimental (non-certified) support for the following flavors of Hadoop. However, the recommended versions are those listed above.

- Pivotal HD 2.0
- Cloudera CDH5 (2.3.0-cdh5.0.0)
- Hortonworks Data Platform 2.0
- Hortonworks Data Platform 2.1

Setting up the HDFS transport application

HDFS Transport application is integrated through RTISH and has the same lifecycle of any other RTISH-managed RTI application.

Here is a basic example of how to configure and start/stop HDFS Transport through RTISH:

1. Create an hdfs-transport application with this command:
   ```bash
tti create application --type hdfs-transport --name hdfs-transport-one
   ```

2. Enable the created hdfs-transport application with this command:
   ```bash
tti enable instance --instances hdfs-transport-one@hdfs-test-host
   ```

3. Start and stop the hdfs-transport application with these commands:
   ```bash
tti start --instances hdfs-transport-one@hdfs-test-host
tti stop --instances hdfs-transport-one@hdfs-test-host
   ```
**Note:** Using RTISH, instances of HDFS transport can be created and started on multiple nodes of a distributed RTI cluster. This is useful when you want to load-balance the data from RTI through multiple hdfs transport instances before writing into HDFS. The more detailed examples below show how to manually configure individual instances to subscribe to only a subset of flow (e.g. “LORS” or “LANDS”). Contact RTI support for additional help with configuring RTI HDFS Transport load-balancing.

### Overriding the HDFS transport configuration property Hadoop Namenode

By default, the Hadoop Namenode URI looks at the location: hdfs://localhost:9000. This can be overridden if the Namenode runs on a different host. For example:

```
rti set property --instances hdfs-transport-one@* --name
   spring.hadoop.fsUri
       --value hdfs://host.domain.com:9000
```

### Overriding default RabbitMQ properties

Similarly, as for AMQP endpoints, the hdfs-transport will look at the following default properties:

- `spring.rabbitmq.host=localhost`
- `spring.rabbitmq.port=5672`
- `spring.rabbitmq.username *`
- `spring.rabbitmq.password *`
- `spring.rabbitmq.virtualHost *`

* - Optional

This can be overridden through RTISH as follows:

```
rti set property --instances hdfs-transport-one@* --name
   spring.rabbitmq.host
       --value host.domain.com

rti set property --instances hdfs-transport-one@* --name
   spring.rabbitmq.port
       --value 5673
```

In the following discussion, note that in HDFS storage, directory paths and “partitions” are basically synonymous. The HDFS directory paths where RTI’s HDFS transport instances place actual data files are comprised of a hierarchical data structure that organizes data files based on their source stream, temporal grouping (or Time Window), and partition sub-grouping. The basic structure is as follows:

```
/<base-path>//<source-stream>//<time-window>//<grouped-partition-id>/
```

where:

- **base-path** is a global user-configurable property (see below)
- **source-stream** is the name of the stream source (for example, “lors”, “lands”, and so on)
Chapter 9: Exporting Data

- **time-window** is the timestamp representing the time bucket for the events, based on the events’ constituent timestamp value.

- **grouped-partition-id** is an integer value between 0 and the configured number of partitions for each named source stream (exclusive). From an HDFS perspective—by adding another sub-directory to the full directory path—the grouped-partition-id provides another level of directory partitioning beyond what’s provided by base-path, source-stream, and time-window. This reduces the chances that worker threads will incur contention while writing data in parallel.

The files written to each directory use the following naming pattern:

```
<source stream>--<source server hostname>--<sequential number in time window>.txt
```

where:

- **source-stream** is the name of the stream source (for example, “lors”, “lands”, and so on)
- **source server hostname** is the hostname of the distribution grid instance that published the events.
- **sequential number in time window** is appended for each new file in the directory. New files are created based on the configurable values for maximum file size and idle timeout.

Following are the detailed descriptions of the RTI properties that control the directory path and file naming:

**base-path**

The base-path is the top-level directory that separates HDFS Transport-produced directories and files from all other (non-RTI produced) data in the HDFS file system.

HDFS transport writes into a set of directories underneath the base-path on the target file system. Here is an example for how to explicitly configure the base-path:

```
rti set property --instances hdfs-transport-one@* --name rti.hdfs.transport.base-path --value /rti-t/data
```

Default value: /rti-t/data

**timestamp-format**

Generate additional subdirectories that are based on a temporal grouping, or time windows, that are defined by the configured timestamp format. Each time a new event arrives with a timestamp that fits into a time window with no prior data (typically indicating the beginning of a new time window), a new subdirectory is created (for example: “/rti-t/data/lors/2017010101/*”).
The HDFS Transport uses Java's SimpleDateFormat to interpret the timestamp format, which provides considerable flexibility. Refer to the JDK doc for detailed options (https://docs.oracle.com/javase/8/docs/api/java/text/SimpleDateFormat.html). The default format for the timestamp is "yyyyMMddHH". This default can be modified for two reasons:

- Select a different time window granularity level.
- Select a different directory name pattern for the same granularity level.

Change granularity to include minute or other levels. Ensure not to make the granularity so small that it creates a large number of directories, which might stress the HDFS namenode metadata management or the underlying file system (the limits vary depending on the actual file system, and so on). Use the format that makes sense for the size of the data. The following is an example for how to explicitly configure the timestamp format. This example adds "HHmm" to the default value and increases the granularity by a factor of 1,440 (60 mins * 24 hrs):

```
rti set property --instances hdfs-transport-one@* --name rti.hdfs.transport.partition.timestamp.format --value yyyymmddHHmm
```

Default value: yyyymmddHH (One Hour time buckets)

### concurrent-consumers

The RTI HDFS Transport allows you to choose the number of concurrent consumers (threads) ingesting the AMQP message streams from the message broker. By default, HDFS transport uses 32 concurrent consumers to consume events from all the stream endpoints in AMQP. The default assumes that HDFS Transport is running on a server class machine handling moderate-to-high event volumes. Based on testing and experience, operators of a deployed RTI system may find that the optimal value might be higher or lower.

Additionally, because it is possible to deploy multiple instances of HDFS Transport to separate (and/or scale) the handling of different event stream flows, the optimal number of concurrent consumers will also depend on the sub-set of message throughput handled by each specific instance. Consult your RTI product Field Engineer or RTI support before tuning the number of concurrent consumers for a production deployment. This example shows how to configure the number of concurrent consumers:

```
rti set property --instances hdfs-transport-one@* --name rti.hdfs.transport.rabbit.concurrent.consumers --value 32
```

Default value: 32

### partitions

To facilitate efficient parallel writing into HDFS, add an additional level, or partition, to the directory tree. Each partition is an integer value between 0 (inclusive) and the configured number of partitions (exclusive), and the partition sub-directory name is the same as the partition integer ID. The number of partitions is configurable for each distinct source stream such as "lores", "lands", and so on.

Each concurrent consumer thread is mapped to a single partition for each named stream. To assure balanced writing across the available partitions, the assigned value for the
partitions property must be a divisible by the number of concurrent consumer threads. Make the value for concurrent consumers and partitions the same to assure that each concurrent consumer thread writes to its own dedicated HDFS partition. When the value for concurrent consumers is a multiple of partitions other than one, then the multiple concurrent consumer threads that map to the same partition ID are called a writer thread group.

The RTI property naming for partitions includes the source stream name, following this pattern: rti.hdfs.transport.<source stream name>.partitions where the source stream name may be a standard output stream such as "lors" or "lands" or a custom- defined metric output stream name. For more information, see Creating user-defined metrics streams.

This example shows how to explicitly configure the number of partitions for the "lors" output stream:

```
rti set property --instances hdfs-transport-one@*  
   --name rti.hdfs.transport.lors.partitions --value 32
```

Default value: 32

**File maximum-size and idle-timeout**

The maximum-size and idle-timeout options control the maximum output file size and the maximum amount of idle time to keep open an HDFS Transport-generated HDFS file before rolling it over to a new one. An internal counter tracks the number of times any specific target file is rolled over (due to either reaching the maximum size or idle timeout) within a time window. The value of this counter is appended to the file name. For example, the first LORS file for hostname, hostname1, will be lors-hostname1-l.txt. After the first rollover, the next filename will be lors-hostname1-2.txt. The Nth filename will be lors-hostname1-N.txt.

The full property names and their default values are (in milliseconds):

- rti.hdfs.transport.file.maximum-size=64M
- rti.hdfs.transport.file.idle-timeout=30000

Using the defaults, files are rolled over if the file size reaches 64 MB or if there is no activity detected in the file buffer for 30 seconds. The maximum file size unit can be designated as kilobytes, megabytes, gigabytes, or terabytes using "K", "M", "G", or "T" as the suffix after the integer number. These examples show how both of these values can be explicitly configured using RTISH:

```
rti set property --instances hdfs-transport-one@*  
   --name rti.hdfs.transport.file.maximum-size --value 64M
```

```
rti set property --instances hdfs-transport-one@*  
   --name rti.hdfs.transport.file.idle-timeout --value 30000
```

Default Values: 64M Maximum Size, 30000 milliseconds idle time.

**Notes on the HDFS file lifecycle**

For all files that HDFS has not closed— for example if the file is still being written—there will be a .tmp extension on the file. This extension is an indication to any downstream
processing applications to not process those files. Otherwise, you might receive stale data or possibly no data at all.

**Note:** The append feature is turned off in the HDFS-transport application as it is in Hadoop. Therefore, the in-progress files have a size of zero and the proper file size is reflected on an HDFS flush and close.

HDFS Transport honors standard UNIX file/directory level permissions. If the user running the application does not have the necessary privileges on the cluster, it will cause security violations. The application itself does not provide any configuration options for access control. Rather, it assumes that the user running the application has full write and execute privileges on the cluster. However, if this is not the case, the Hadoop platform must be configured so that the user running the hdfs transport sub-system is part of a proper Local Access Control List (ACL) that can perform the necessary operations (execute and write). ACL is a feature that was added in 2.4.0 version of Hadoop and is an HDFS admin level operation. The hdfs transport sub-system user must be added to a UNIX group, which must be added to the ACL on HDFS.

If you use the ACL approach, consult with your HDFS administrator to configure a directory with write and execute permissions for your user. This same directory path must then be used as the value for the property `rti.hdfs.transport.base-path`.

The following is a possible sequence of steps to follow when using ACLs. The directory "/rti-t/data" is only an example, although it is the default path, assuming that your user id is "rti" and there is an OS group called "rti-ops".

1. Work with HDFS admins to create a directory named /rti-t/data on HDFS:
   ```
   bin/hdfs dfs -mkdir /rti-t
   bin/hdfs dfs -mkdir /rti-t/data
   ```

2. Make this directory read, write, execute by any user in the rti-ops group:
   ```
   bin/hdfs dfs -setfacl -m group:rti-ops:rwx /rti-t/data
   ```

3. Ensure that this same directory is set for `rti.hdfs.transport.base-path`. (Change the default /rti-t/data to something that is custom). In RTISH, execute this command:
   ```
   rti set property --instances hdfs-transport-one@hdfs-test-host --name rti.hdfs.transport.base-path --value /rti-t/data
   ```

For a tutorial on how ACLs can be enabled on the Hadoop cluster, refer to:


**Note:** Currently, the HDFS Transport application does not support Kerberized Hadoop environments. The assumption currently is that the end user has access to a non-Kerberized HDFS cluster. Support for Kerberos security-enabled HDFS target platforms will be considered in the future.

**From the metrics streams**

Set the following property to true to consume metrics stream in HDFS transport:
rti set property --instances hdfs-transport-one@*
   --name rti.hdfs.transport.stream.metrics.enabled --value true
rti set property --instances hdfs-transport-one@*
   --name rti.hdfs.transport.metrics.streams.routing.keys
   --value celtower,device,mobilenetwork,subscribergroup

When this property is enabled, the desired metrics stream names must be given to the application:

**Note:** The metrics stream names must match what is configured on the distribution grid. This might be one of the four standard/out-of-box metric streams (celltower, device, mobilenetwork, subscribergroup) (see Metrics streams), or a user-defined metric stream (see Creating user-defined metrics streams). If more than one metric stream is specified, they must be comma-separated.

The default number of partition groups for metrics data writing in HDFS is four. The number of partitions can be explicitly configured using the following:

rti set property --instances hdfs-transport-one@*
   --name rti.hdfs.transport.metrics.partitions --value 4

Here is a sample output file name for the metrics data stream in HDFS:

/rti-t/data/metrics/celltower/2014122614/
host.domain.com_celltower-metrics-events-stream-0.txt

Note that the path itself contains patterns like metrics/[stream name] and then again as part of the file name pattern [server host name]_[stream name]-metrics-evens-stream-[x].txt

**Note about default metrics behavior:** RTI generates metrics data for AMQP as compressed gzip data and as such when HDFS Transport consumes it, the data cannot be processed as normal csv text data. Whenever hdfs-transport consumes non-textual data (including non-metrics streams such as LORS, LANDS, and ESS provided that they were written into AMQP as compressed data), it encodes each byte of the payload as a hexadecimal string and appends it with all the AMQP headers. A template would be as follows:

Hex encoded string,key1:val1,key2:val2,...,keyN:valN

The charset used for encoding is UTF-8, which is the encoding used by RTI for compressing CSV output. Use this charset when decoding this data.

From that point onward, this data can be treated as plain text for downstream processors. It is up to the end user to decode this data properly for their use cases.

**Consuming LANDS events**

Exporting LANDS data through HDFS transport is similar to the LORS flow, except that it is not enabled by default. To enable LANDS, use the following:

rti set property --instances hdfs-transport-one@*
   --name rti.hdfs.transport.stream.metrics.enabled
   --value true
When enabled, LANDS data can be found at the following path (as an example):

/rti-t/data/lands/2014122615/hostname1_lands-events-stream-0.txt

The default number of LANDS partition groups is 4. It can be explicitly configured as follows:

```bash
rti set property --instances hdfs-transport-one*@ --name rti.hdfs.transport.lands.partitions
```

**Consuming ESS events**

The consumption of ESS data in HDFS is similar to the metrics consumption. It must be enabled explicitly:

```bash
rti set property --instances hdfs-transport-one*@ --name rti.hdfs.transport.stream.ess.enabled
```

Also, the ESS IDs, qualified with the Org/OrgUnit owners of the ESS IDs, must be provided to the HDFS Transport application explicitly. The Org, OU, and ESS ID are combined using "." (dot) notation using the pattern orgName.ouName.subscriptionId. To specify a set of multiple Event Stream Subscriptions, use a comma-separated list.

For example, if Org “ABC” and OU “123” has ESS IDs “SUB01”, “SUB02”, and “SUB03”, use the following:

```bash
rti.hdfs.transport.ess.stream.subscriptions=ABC.123.SUB01,ABC.123.SUB02,ABC.123.SUB03
```

The combinations of values provided must match ESS subscriptions provisioned to the distribution grid.

The default number of ESS partition groups is four. It can be explicitly configured as follows:

```bash
rti set property --instances hdfs-transport-one*@ --name rti.hdfs.transport.ess.partitions
```

An example of an ESS output in HDFS might look like the following:

```
/rti-t/data/ess/ABC.123.SUB05/2014122613/hostname1_ABC.123.SUB05-ess-events-stream-0.txt
```
Using the Gzip compression codec for HDFS output

The compression support in HDFS Transport allows the output data to be compressed using the Gzip codec. This can be enabled by setting the following property:

```
rti set property --instances hdfs-transport-one@*
    --name rti.hdfs.transport.compression.codec
    --value gzip
```

This would apply gzip compression to all the data that is being written to HDFS. Here is an example of compressed output file:

```
/rti-t/data/metrics/device/2014122614/host.domain.com-3/device-metrics-events-stream-1.txt.gz
```

The GZIP codec does not allow splitting the file in HDFS for downstream processors. The whole file needs to be extracted and consumed.

**Note:** The GZIP codec makes exporting data in HDFS slower than uncompressed data. If the Gzip compression codec is selected, consider compensating by increasing the number of concurrent consumers that process data from AMQP.

Using other compression codecs supported by HDFS-Transport

There are other faster, splittable and non-splittable compression codecs that you can choose if the target HDFS platform supports those compression codecs. Note that these codecs cause runtime errors if these compression codecs are not supported by the target Hadoop.

**Note:** All these codecs are currently supported in HDFS Transport only on an experimental basis. Exercise caution if you enable any of the following listed compression codecs.

The following are examples of how to enable these other compression codecs:

- **Snappy** - Non-splittable:
  
  ```
  rti set property --instances hdfs-transport-one@*
      --name rti.hdfs.transport.compression.codec
      --value snappy
  ```

- **Bzip2** - Splittable:
  
  ```
  rti set property --instances hdfs-transport-one@*
      --name rti.hdfs.transport.compression.codec
      --value bzip2
  ```

- **LZO** - Non-splittable:
  
  ```
  rti set property --instances hdfs-transport-one@*
      --name rti.hdfs.transport.compression.codec
      --value lzo
  ```

- **SLZO** - Splittable:
Enabling JMX support

JMX can be enabled for hdfs-transport application as in any other RTI application. All metrics will be available under the domain io.pivotal.rti.data.hdfs.transport. Here is an example using port 1099.


Use tools such as VisualVM or JConsole to examine the status of the HDFS export. Here are the supported operations:

General metrics:

- **getAllAvailableProperties**—List all the configuration properties available to HDFS Transport.
- **getApplicationSpecificProperties**—List all the application level configuration properties.

For LORS and LANDS

- getMessageCountTransportedToHdfs
- getHdfsWriteRatePerSecond

For metrics

- **getEventsTransportedToHdfs**—List all the enabled metrics streams in the form of key/value pairs where key is the stream name and value is the number of metrics written to HDFS for that stream so far.

For ESS

- **getEventsTransportedToHdfs**—Key/value pair where key is the ESS and value is the number of events written to HDFS for that ESS so far.
- **getHdfsWriteRatePerSecond**—Key/value pair where key is the ESS and value is the rate of ESS export into HDFS per second for that stream.
Note: The per-second rate calculations are based on rough estimates. They give you an average rate of HDFS writes per second between the first and last writes.

Logging
Log files are available in the standard RTI locations. All system and application properties used by the HDFS Transport are logged on startup. Each minute, the log files report the total number of events written to HDFS for each stream. This duration can be configured using the following properties:

\[
\begin{align*}
\text{rti.hdfs.transport.lors.count.log.duration} \\
\text{rti.hdfs.transport.lands.count.log.duration} \\
\text{rti.hdfs.transport.metrics.count.log.duration} \\
\text{rti.hdfs.transport.ess.count.log.duration}
\end{align*}
\]

Miscellaneous properties
The default exchange used by RTI for sending LORS, LANDS and ESS data is DECODER.STREAM. If your RTI deployment does not use this exchange, set it explicitly in hdfs-transport using the following property:

\[
\text{rti.hdfs.transport.default.exchange}
\]

The default routing keys for LORS and LANDS are DECODER.STREAM.ALL.LORS and DECODER.STREAM.ALL.LANDS respectively. If they are not the ones used in RTI, they must be explicitly stated in hdfs-transport using the following properties:

\[
\begin{align*}
\text{rti.hdfs.transport.lors.routingKey} \\
\text{rti.hdfs.transport.lands.routingKey}
\end{align*}
\]

RTI has can write binary data in either raw or hexadecimal encoded format. By default, RTI writes the binary version of a message to HDFS. To enable RTI to convert the binary message to hexadecimal encoded format, set this property to true:

\[
\text{rti.hdfs.transport.encode=true}
\]
InfluxDB transport

The InfluxDB transport is an application that runs outside of RTI. It listens for messages coming through AMQP, transforms them into InfluxDB messages, and then sends them to the Influx database. Custom message transformers can be written using Groovy and registered in a JSON config file. The InfluxDB transport is typically used with a Grafana dashboard UI.

Note: The Grafana and InfluxDB products are not supported by EMC. They are used to show an example of how to visualize the RTI output.

Configuration

All InfluxDB scripts and configuration files can be found in the RTI distribution under tools/influxdb. The etc/config.json file contains all the configuration details for the message transformers.

An example config.json with two transformers:

```
[{
   "name" : "metricTransformer",
   "enabled" : true,
   "exchange" : "decoder.metrics",
   "routingKey" : "*",
   "scriptName" : "metricsTransformer.groovy",
   "databaseName" : "rti",
   "databaseAddress" : "localhost:8086"
 },{
   "name" : "alertTransformer",
   "enabled" : true,
   "exchange" : "decoder.stream",
   "routingKey" : "decoder.stream.qos.alert",
   "scriptName" : "alertTransformer.groovy",
   "databaseName" : "rti",
   "databaseAddress" : "localhost:8086"
 }]
```

Transformer properties

- name - A unique name for the transformer
- enabled – Indicates whether a transformer will process events
- exchange - The AMQP exchange to listen to
- routingKey - The routing key for AMQP
- scriptName - The Groovy script that transforms the message
- databaseName - The Influx database in which to insert the message
- databaseAddress - The address of the Influx database

Groovy script

The Groovy script must have a method named “transform” that accepts an AMQP message and returns a string.
The following is an example Groovy transformer:

```java
import org.springframework.amqp.core.Message

def transform(Message message){
    String alertMessage = new String(message.getBody());
    StringBuilder strBuilder = new StringBuilder();
    strBuilder.append("qos_alert,message=");
    strBuilder.append(alertMessage.replaceAll(" ", "\\ \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\
    \\```n

The returned String will be sent to the InfluxDB defined by the transformer's database properties. The format of the string must follow InfluxDB's line protocol. For more information about this format, refer to the InfluxDB documentation: https://docs.influxdata.com/influxdb/v0.9/write_protocols/line/

Example string sent to InfluxDB:

cpu,host=server01,region=uswest value=1

To start the application, run bin/influxdb-transport. If startup is successful, a message for each transformer that has been started appears.

Example message of two transformers successfully started:

```
2016-06-14 14:38:04,678 [main] INFO io.pivotal.rti.influxdb.InfluxDBTransport - metricTransformer started, bound to queue: amq.gen-DVoOAJ6P6VRefYFiqhWmYA
2016-06-14 14:38:04,683 [main] INFO io.pivotal.rti.influxdb.InfluxDBTransport - alertTransformer started, bound to queue: amq.gen-x2Fcl_2LFd45k9ydGkHqUw
```
Chapter 10 Managing the RTI System

This chapter presents the following topics:

Managing the EMC RTI System .............................................................. 158
Monitoring IG to DG connectivity and flow ........................................... 166
Restarting the system ........................................................................... 168
Modifying configuration properties ....................................................... 172
Backup and restore guidelines ............................................................... 172
Recovering from a system failure ......................................................... 179
RTI error messages ............................................................................. 182
RTI log files ....................................................................................... 185
Managing the RTI System

This section covers tasks that administrators might need to perform to check the health of the RTI system and troubleshoot problems:

- Checking system health
- Backing up and restoring data
- Safeguarding the RTI configuration

Checking system health

When you have installed and configured an operational RTI system that is capable of ingesting and processing output, you might need to check its state and health periodically. In general, RTI is a self-maintaining and self-healing system. This section explains how to ensure that the system is fully operational and healthy.

The state and health of an RTI installation applies to two different levels:

- System level
- Application instance level

You can use the `rti status` command to check system or application state (what is running or not running in the system).

The expected list of processes that need to be running in an operational RTI system depends on your configuration. At a minimum, an operational system will have application instances and associated processes for the following applications:

- Ingesters
- Ingest grids
- Distribution grids
- Locators
- Provisioning Manager
- RabbitMQ broker

Other optional processes may be part of your system, including the REST client. Note that deployments spanning different data centers might be separately deployed and have only a subset of these applications (on each side of the WAN replication link).

Monitoring the system health is a broader task, with both resource and performance implications. The health of the system depends on both resource limitations (disk space, memory, cluster size, and so on) and performance tuning to optimize throughput.

Monitoring the system

You can check event activity at various points in the system and monitor critical system attributes. For example, you can return counts of the number of events that are in process or were discarded at a given point in the data flow. The following sections describe the statistics you can collect for different RTI components.
Examples of the `rti show metrics` command are installed with RTI in the `docs/examples/showAllMetrics.cmd` file.

**Using the Show Metrics command**

The RTISH command-line interface provides an `rti show metrics` command, which returns RTI metrics based on JSON path queries. Alternatively, you can access the same metrics by using a third-party JMX monitoring tool such as VisualVM. VisualVM is available for download at [https://visualvm.github.io/](https://visualvm.github.io/). You can use the generic JMX API programmatically.

The `rti show metrics` command returns all of the JMX-monitored metrics (MBean attribute names and values) for specific RTI processes. The command takes two parameters: the process ID of the RTI process you want to monitor and a JSONPath query string that constrains the JSON output. For example, to return all of the available metrics for a specific process ID, enter a command like this:

```
rti show metrics --process <pid> --all --query $ 
```

For details about the JSONPath query language, refer to:

- [http://goessner.net/articles/JsonPath/](http://goessner.net/articles/JsonPath/)

**Using VisualVM for monitoring**

The same metrics that you can query with the `rti show metrics` command are available with UI tools such as VisualVM. To set up and use VisualVM, follow these steps:

1. Download and install VisualVM on an RTI host.
3. Map the process IDs to specific RTI applications by checking the results of the `rti list processes` command.
4. Open the process ID for the application you want to monitor (such as an Ingester Grid process or a distribution grid process).
5. Select the MBeans browser.
6. Open the MBean in which you are interested. For example, if you are interested in the number of events flowing out of the Ingester:
   ```
io.pivotal.rti.ingestgrid/MessageChannel/fromIngestor
   ```
7. Look for the MBean attribute in which you are interested. For example:
   ```
   SendCount
   ```

The following screen shows the `SendCount` value (860992) and other attributes for events flowing from the ingester into the ingest grid:
You can also use VisualVM to enable and disable protocols in the ingester. Go to `io.pivotal.rti.ingester`, select **Protocol Manager**, then select **Operations**. The following example shows how to enable the iucs protocol.

![Figure 16. SendCount value and other attributes](image)
You can monitor the end-to-end flow of events through the system and compare counts at different endpoints:

- Number of events that flow into the ingester
- Number of events that flow out of the ingester
- Number of events that flow into the ingest grid
- Number of events that flow out of the ingest grid
- Number of events that flow into the distribution grid
- Number of events that flow out of the distribution grid into:
  - AMQP format (RabbitMQ)
  - Output files

For example, you can verify that the count of events that flow out of the ingester matches the count of events that flow into the ingest grid. A first level of monitoring includes tracking and matching counts. When you know what the counts are, you can investigate further to understand why the counts at given endpoints do not match. You can look for events that were dropped; for example, issues may arise with throughput overloads or network traffic between nodes. You can also understand why certain events were filtered out.
For reference, the following examples list the domain, MBean type, and MBean name for each query.

**Events flowing into the ingester**

The number of events that flow into the ingester is the `EventCount`, which is tracked per protocol. To find the total count of records flowing into the ingester, calculate the sum of the `EventCount` values for the enabled protocols. (First, use `rti list processes` to return the process ID for the ingester application.)

**Table 21. Ingestor incoming event count MBeans**

<table>
<thead>
<tr>
<th>Domain</th>
<th>io.pivotal.rti.ingester.&lt;protocolname&gt;.source</th>
</tr>
</thead>
<tbody>
<tr>
<td>MBean type</td>
<td>&lt;registered-protocol-adapter-name&gt;</td>
</tr>
<tr>
<td>MBean name</td>
<td>&lt;protocolname&gt;ProtocolAdapter</td>
</tr>
</tbody>
</table>

To show metrics associated to a particular protocol adapter, use this command:

```
rti show metrics --process <ingester-pid> --query .[?(@.name.name=='<protocol-adapter-name>')]```

Where `<protocol-adapter-name>` is one of:

- `adrProtocolAdapter`
- `csvProtocolAdapter`
- `gbProtocolAdapter`
- `datacastGngiProtocolAdapter`
- `isupProtocolAdapter`
- `iucsProtocolAdapter`
- `iucslupsProtocolAdapter`
- `iupsProtocolAdapter`
- `jsonProtocolAdapter`
- `radiusProtocolAdapter`
- `datacastS1mmeProtocolAdapter`
- `tdrProtocolAdapter`
To show metrics associated with all protocol adapters, follow the example below:

```
rtish> rti show metrics --process <ingester-pid> --query .[?(@.attr.ProtocolName)]
[
  "name" : {
    "name" : "isupProtocolAdapter",
    "domain" : "io.pivotal.rti.ingester.isup.source",
    "type" : "IsupProtocolAdapter"
  },
  "attr" : {
    "ExtendedParsers" : {
      "3gpp-ie-dataid-based-parsers" : [ ],
      "tek-ie-bit-positional-parsers" : [ ],
      "tek-ie-dataid-based-parsers" : [ ]
    },
    "ErrorCount" : 0,
    "ExtendedParsingEnabled" : true,
    "EventCount" : 0,
    "LastEventTime" : 0,
    "ProtocolName" : "isup",
    "Version" : "6.15.2",
    "LastErrorTime" : 0
  }
}, {
  "name" : {
    "name" : "adrProtocolAdapter",
    "domain" : "io.pivotal.rti.ingester.adr.source",
    "type" : "AProtocolAdapter"
  },
  "attr" : {
    "ExtendedParsers" : {
      "3gpp-ie-dataid-based-parsers" : [ ],
      "tek-ie-bit-positional-parsers" : [ ],
      "tek-ie-dataid-based-parsers" : [ ]
    },
    "ErrorCount" : 0,
    "ExtendedParsingEnabled" : true,
    "EventCount" : 860992,
    ...]
```

**Events flowing out of the ingester**

The total number of events that flow out of the ingester for a specified adapter is the HandleCount value for the adapter’s processor.

**Table 22. Ingester outgoing event count MBeans**
Domain | Domain | io.pivotal.rti.ingester.<protocolname>.processor
---|---|---
MBean type | MessageHandler | MessageChannel
MBean name | gemfireClient | input

For example:

```
rtish> rti show metrics --process <ingester-pid> --all --query .[?[@.name.name='gemfireClient']].[?[@.name.domain='io.pivotal.rti.ingester.adr.processor']].attr.HandleCount
```

This value is also a per-protocol value (adr in this example).

**Events flowing into the ingest grid**

The total number of events that flow into the ingest grid is the `SendCount` value for the ingest grid process.

Table 23. Ingest grid incoming event counter MBeans

| Domain | Domain | io.pivotal.rti.ingestgrid.processor
---|---|---
| MBean type | MessageChannel | MessageChannel
| MBean name | input | input

For example:

```
rti show metrics --process <ingest-grid-pid> --query .[?[@.name.type='MessageChannel']].[?[@.name.name='input']].attr.SendCount
```

**Events flowing out of the ingest grid**

The total number of events that flow out of the ingest grid is the `HandleCount` value for the ingest grid process.

Table 24. Ingest grid outgoing event counter MBeans

| Domain | Domain | io.pivotal.rti.ingestgrid.processor
---|---|---
| MBean type | MessageHandler | MessageHandler
| MBean name | outboundAdapter | outboundAdapter

For example:

```
rti show metrics --process <ingest-grid-pid> --all --query .[?[@.name.name='outboundAdapter']].attr.HandleCount
```
Events flowing into the distribution grid (from the WAN Gateway)
Show the number of events flowing into the distribution grid from the WAN Gateway.

Table 25. Distribution grid incoming event counter MBeans

<table>
<thead>
<tr>
<th>Domain</th>
<th>io.pivotal.rti.distgrid.processor</th>
</tr>
</thead>
<tbody>
<tr>
<td>MBean type</td>
<td>MessageChannel</td>
</tr>
<tr>
<td>MBean name</td>
<td>defaultStream</td>
</tr>
</tbody>
</table>

For example:
```
rtish> rti show metrics --process <dist-grid-pid> --query .[?(@.name.type=='MessageChannel')].[?(@.name.name=='defaultStream')].attr.SendCount
```

Events flowing out of the distribution grid: AMQP Format
Return the total number of location events that flow out of the distribution grid (in AMQP message format).

Table 26. Distribution grid outgoing location AMQP event counter MBeans

<table>
<thead>
<tr>
<th>Domain</th>
<th>io.pivotal.rti.distgrid.processor</th>
</tr>
</thead>
<tbody>
<tr>
<td>MBean type</td>
<td>MessageHandler</td>
</tr>
<tr>
<td>MBean name</td>
<td>locationFeedEventsOutboundChannelAdapter</td>
</tr>
</tbody>
</table>

For example:
```
rtish> rti show metrics --process <dist-grid-pid> --all --query .[?(@.name.name=='locationFeedEventsOutboundChannelAdapter')].attr.HandleCount
```

Return the total number of transaction events that flow out of the distribution grid (in AMQP message format).

Table 27. Distribution grid outgoing transaction AMQP event counter MBeans

<table>
<thead>
<tr>
<th>Domain</th>
<th>io.pivotal.rti.distgrid.processor</th>
</tr>
</thead>
<tbody>
<tr>
<td>MBean type</td>
<td>MessageHandler</td>
</tr>
<tr>
<td>MBean name</td>
<td>transactionFeedEventsOutboundChannelAdapter</td>
</tr>
</tbody>
</table>

For example:
```
rtish> rti show metrics --process <dist-grid-pid> --all --query .[?(@.name.name=='transactionFeedEventsOutboundChannelAdapter')].attr.HandleCount
```
The AMQP event counts represent the union of all events that have been subscribed to using ESS messages.

**Events flowing out of the distribution grid: output files**

To return the number of events that go to the location and transaction output files, run the following queries:

- The total number of location events that flow out of the distribution grid (to file):

  ```
  rtish> rti show metrics --process <dist-grid-pid> --all --query .?[(@.name.name=='locationFileOutputChannelAdapter')].attr.HandleCount
  ```

  **Table 28. Distribution grid outgoing location file event counter MBeans**

<table>
<thead>
<tr>
<th>Domain</th>
<th>io.pivotal.rti.distgrid.processor</th>
</tr>
</thead>
<tbody>
<tr>
<td>MBean type</td>
<td>MessageHandler</td>
</tr>
<tr>
<td>MBean name</td>
<td>locationFileOutputChannelAdapter</td>
</tr>
</tbody>
</table>

- The total number of transaction events that flow out of the distribution grid (to file):

  ```
  rtish> rti show metrics --process <dist-grid-pid> --all --query .?[(@.name.name=='transactionFileOutputChannelAdapter')].attr.HandleCount
  ```

  **Table 29. Distribution grid outgoing transaction file event counter MBeans**

<table>
<thead>
<tr>
<th>Domain</th>
<th>io.pivotal.rti.distgrid.processor</th>
</tr>
</thead>
<tbody>
<tr>
<td>MBean type</td>
<td>MessageHandler</td>
</tr>
<tr>
<td>MBean name</td>
<td>transactionFileOutputChannelAdapter</td>
</tr>
</tbody>
</table>

The events going to file must be equivalent to all the events that pass the transaction opt-in.

**Monitoring IG to DG connectivity and flow**

RTI includes JMX MBeans to monitor the status of ingest grid (IG) to distribution grid (DG) connectivity and associated per-destination transmission queues. The MBean control package name is com.pivotal.rti.gemfire.wan, and the MBean name is "SingleHopGateway". The MBean has the following available attributes:

- **BatchSize**: Displays the configured maximum batch size (property name "rti.ingestgrid.gateway-sender.batch-size").
- **BatchInterval**: Displays the configured maximum batch interval (property name "rti.ingestgrid.gateway-sender.batch-time-interval").
• **ConcurrencyLevel**: Displays the configured number of concurrent dispatcher threads and queues (property name "rti.ingestgrid.single-hop-gateway-sender.dispatcher-threads").

• **Connected**: Indicates whether the single-hop gateway is connected to the distribution grid, by using either "True" or "False".

• **ConnectionStatus**: Provides a way to determine if reconnection is occurring due to network or downstream distribution grid problems. Valid values are "Connected", "Reconnecting", or "Not Connected".

• **CurrentQueueSizes**: Contains a list of the remote distribution grid connections, and the size of the outbound transmission queue for each one. This string’s format is "Name=<IG Instance Name>@<IG hostname>_DG hostname_<DG Port>,Size=<queue size>;<additional entries for each connected remote DG server>..."

  For example:

  Name=IngestGrid1@Sclf041.lss.emc.com_172.28.12.6_20022,Size=0;Name =IngestGrid1@Sclf041.lss.emc.com_172.28.12.15_20022,Size=0;

• **MaxQueueSize**: Determines the current maximum queue size for any of the transmission queues. If this number rises substantially beyond the BatchSize, then at least one of the remote distribution grid (DG) servers is not consuming events fast enough to keep up with its allocated event flow. This issue might be caused by a problem on the DG host, a problem with the DG output not being able to keep up, or a problem in the network between the IG and DG. You can find per-DG instance maximum queue sizes by inspecting the CurrentQueueSizes attribute. Monitor this attribute at all times to find any throughput problems as soon as they happen. Typically, MaxQueueSize cannot be more than three times the configured BatchSize for more than a few seconds at a time. A consistently growing MaxQueueSize indicates that the system is at risk of crashing.

• **RemoteConnectionNames**: Lists the remote distribution grid instances, and their connectivity information. This string’s format is "RemoteServerLocation [hostname=<hostname or IP address>, hostnameForSender=<DG configured property "rti.gateway-receiver.hostname-for-senders"> port=<DG configured property "rti.single-hop.gateway.hub.port"> cacheServerPort=<The dynamically assigned Cache Server port for the remote DG>];<additional entries for each connected remote DG server>..."

  For example:

Chapter 10: Managing the RTI System

Restarting the system

Restarting the RTI system is a set of administrative actions that take a normally running cluster offline and subsequently bring it back online. The procedures covered in this section are specific to restarting the system in a controlled way. The restart process is different from any emergency or abnormal intervention you might take.

RTI supports a clean restart of the whole system, along with clean independent restarts of individual applications:

- A cold restart procedure is a complete restart of all of the running processes and associated brokers that belong to the RTI system.
- A hot restart procedure does not take RTI completely offline. Specific components are stopped and restarted independently.

Cold restart

The inventory for a cold restart includes processes for all of the following applications and brokers:

- Ingesters
- Ingest grid nodes
- Ingest grid locators
- Distribution grid nodes
- Distribution grid locators
- Provisioning Manager
- Rabbit MQ broker
- REST server

The restart procedure takes the client processes (ingesters and Provisioning Manager) offline, followed by the grid nodes, locators, and the Rabbit MQ broker. All of the processes are stopped using the RTISH administration tool. For information about the status, stop, and start commands, refer to RTISH commands.

The sequence of operations for a cold restart is as follows:

1. Stop the workflow:
   a. All ingesters are stopped.
   b. Ingest grid REST client is stopped.
   c. All of the ingest grid nodes are stopped.
   d. All of the ingest grid locators are stopped.
   e. Provisioning Manager is stopped.
   f. Distribution grid REST client is stopped.
   g. All distribution grid nodes are stopped.
   h. All distribution grid locators are stopped.
   i. Rabbit MQ broker is stopped.
2. Perform any offline intervention and maintenance operations (for example, diskstore compaction, offline backups, node maintenance, application of patches to external software, or the operating system).

3. Restart the workflow:
   a. Start the Rabbit MQ Broker.
   b. Start all distribution grid locators.
   c. Start all distribution grid nodes.
   d. Start the distribution grid REST client.
   e. Start the ingest grid locators.
   f. Start the ingest grid nodes.
   g. Start the ingest grid REST client.
   h. Start the Provisioning Manager.
   i. Start all ingesters.

The REST clients are optional but must be stopped and restarted if they are in use. For example, the following commands stop an ingester grid:

```
// Ordered instance stop MTX
//Clients first
rti stop --instances ingester1@*
rti stop --instances ingest-server4@*,ingest-server3@*,ingest-server2@*,ingest-server1@
sleep --delay 3000
rti stop --instances ingest-locator1@
```

The following commands stop a distribution grid:

```
// Ordered instance stop IT DC
//Clients first
rti stop --instances dist-provisioning@
rti stop --instances dist-server3@*,dist-server2@*,dist-server1@
sleep --delay 3000
```

The following commands start a distribution grid:

```
// Ordered instance start:
rti start --instances dist-locator1@
sleep --delay 3000
rti start --instances dist-server1@
rti start --instances dist-server2@*,dist-server3@
sleep --delay 15000
rti start --instances dist-provisioning@
```

The following commands start an ingester grid:

```
// Ordered instance start MTX
rti start --instances ingest-locator1@
```
Restarting specific components in RTI is possible with some constraints. The stop and start procedures are no different than those explained in Chapter 16 Referencing RTI Commands. This section clarifies the constraints that arise when you restart specific processes of the RTI cluster and the associated impact to the event flow and processing. Note that the “rti system shutdown” command always applies to all defined started or enabled instances and is not used to control specific individual sub-components.

### Restarting the ingester

You can restart the ingester component in isolation with no impact to other RTI components. However, the restart results in an interruption to the event flow of the protocol streams that the ingester handles. When RTI is deployed in a highly available configuration, the interruption to the event flow can be minimized based on the capabilities of an external load balancer (not supplied with RTI) and the ability of upstream event sources to reconnect to the other ingester instances. The ingester process requires at least one ingest grid node to be online prior to start. For example, here are the status, stop, and start commands for a hypothetical ingester:

```
rti status --instances ingester1@
rti stop --instances ingester1@
rti start --instances ingest-locator1@
```

### Restarting ingest grid and distribution grid nodes

The grid nodes can be restarted independently of each other and of the other processes in the cluster. RTI is elastic and automatically scales up and down to accommodate the event stream flows to the remaining nodes that are online. However, on restart, some data loss is possible for inflight events on the node that is restarted if the grid is not configured with data redundancy. The data redundancy level is configured using the `rti.ingestgrid.pr.redundancy` or `rti.distgrid.feed.pr.redundancy` property. The default is zero (no redundancy). When the process comes up again, the grid automatically scales to rebalance event flows to the new node. For production systems, the redundancy must never be set to 0 because this prevents automated failover from working.

The nodes can also be taken down in groups of more than one if there is enough capacity to handle the flowing traffic on the remaining nodes. However, the number of nodes taken down at a time must never exceed the configured redundancy level, or all copies (both the Primary and the Secondary) for some of the bucket partitions will be lost. Manual intervention is required to revoke the missing partition’s GemFire disk stores.

Configuring a high redundancy level allows faster system rolling restarts. A restart of the grid nodes is expected to cause a brief drop in the overall system throughput commensurate with the number of nodes that are being restarted. If the number of nodes to be restarted will leave insufficient capacity to process event stream flows, you must

```
sleep --delay 3000
rti start --instances ingest-server1@
rti start --instances ingest-server2@*,ingest-server3@*,ingest-server4@
sleep --delay 3000
rti start --instances ingest-server1@
```
disable incoming flows to reduce the traffic volume to tolerable levels, or choose a less busy time.

To take all ingest grid nodes offline, all the ingester processes must go offline or, at a minimum, have all the event flows disabled (they are clients to the grid nodes). If all the distribution grid nodes are taken offline, the Provisioning Manager must be taken offline.

When ingester/distribution grid nodes are restarted, there might be a temporary loss in the reference data stored in memory, depending on the configured data redundancy level. However, when data persistence is enabled, the data is restored from the persisted disk stores.

For example, here are the status, stop, and start commands for two ingest grid nodes:

```bash
rti status --instances ingest-server3@*,ingest-server2@*
rti stop --instances ingest-server3@*,ingest-server2@*
rti start --instances ingest-server2@*,ingest-server3@*
```

**Ingest grid and distribution grid locators**

The locator processes can be restarted in isolation to the other processes. When the locators are restarted, they re-establish discovery amongst themselves. EMC recommends having at least two locator processes, or at a minimum, one available at all times. If not all locators are available, the system continues to function but no other processes (grid nodes, ingester, or Provisioning Manager) can be restarted during that period, and clients’ application connectivity can degrade. The RTISH tool grid commands, Reference Data Loader (RDL) commands, and the optional Pulse Dashboard are not available when no locators are available online.

For example, here are stop and start commands for a locator:

```bash
rti stop --instances dist-locator1@*
rti start --instances dist-locator1@*
```

**Provisioning manager**

The Provisioning Manager (PM) can be restarted in isolation from other RTI-T processes. The PM is a client to the distribution grid and the Rabbit MQ broker, implying that there needs to be at least one distribution node and a Rabbit MQ broker available before starting the PM. When the PM is offline, external provisioning requests are not consumed and processed. However, as long as there is capacity in the Rabbit MQ broker queues, the messages are held there and the PM resumes processing them in the order they were submitted when it comes back online. Any external provisioning clients that must process responses from the PM experience delays in obtaining response messages during the intervention.

For example, here are stop and start commands for a PM:

```bash
rti stop --instances dist-provisioning@*
rti start --instances dist-provisioning@*
```

**Rabbit MQ broker**

The Rabbit MQ broker can be restarted in isolation from other RTI-T components.
However, when the Rabbit MQ broker is offline, there is likely to be data loss in the event streams over AMQP if the broker is not in a clustered configuration. External provisioning clients cannot submit messages to RTI during that time. The Rabbit MQ restart and administration operations are covered separately in the Rabbit MQ documentation.

**Tools**

All of the RTI tools (RTISH, RDL, REST client, and grid dashboard) can be stopped and restarted independently from all other components as long as there is at least one grid locator and one grid node online.

**Modifying configuration properties**

All configuration changes require a restart of the application or the system. You have to stop the application, reset properties using RTISH commands, and then restart the application.

*Caution:* Do not update properties directly in the configuration files. All instance-specific properties are stored in a master cluster configuration file (automatically backed-up to all nodes). Regenerate the deployed instance configuration from the master.

There is a `defaults-properties` file for each application type and an associated `config-properties` file; the changes you make using RTISH result in updates to the `config-properties` file.

*Note:* Whenever you change your system configuration, be sure to record the changes in a master script and back it up.

**Backup and restore guidelines**

This section explains how to safeguard the RTI system. You must protect the following assets:

- Reference data
- System configuration scripts

**Backing up and restoring RTI data**

RTI reference data is stored in GemFire regions, which automatically provide a level of redundancy. The loaded data is automatically stored on disk. Data that is replicated to all nodes is stored to disk on all nodes. Data that is partitioned across nodes is stored to disk in one or more locations, according to the configured redundancy level. If you lose a copy of the data because a server fails, the system continues to operate. However, a catastrophic failure that involves multiple servers or the entire cluster results in data loss and the need to recover the system's reference data from a backup source. This source may be the original source files loaded using RDL or the persistence files described in the following procedure.

You may also be interested in setting up a new RTI environment or hardware based on an existing system; in this scenario, you also need to take backups of an existing system prior to moving the system or re-creating it.
Usually, with RTI you back up the reference data because this data is the core persistent data that the system needs to function. However, it is easy to reload this data from the original set of files. In any case, always maintain backups of these load files. Data that is provisioned into RTI from third-party applications (event subscriptions) must be backed up independently by those third parties. Data that moves through the system in event streams is considered ephemeral and is not backed up to disk. The system configuration is deployed on all hosts using RTISH. You can redeploy it, if needed, by running the same scripts that you used to create the system initially.

1. During a period of inactivity, bring the whole system down by running the `rti system shutdown` command. This operation ensures that the GemFire persistence files are closed cleanly, reducing startup time when the system is restarted.

2. Take backups of the following directories on your ingest grid and distribution grid servers:
   - `~/emc_rti_<version>/var/data/<grid_instance_name>/diskstore` (This is the most important directory to back up. It contains persisted data for all of the regions.)
   - `~/emc_rti_<version>/var/data/<grid_instance_name>/gateway`
   - `~/emc_rti_<version>/var/data/<grid_instance_name>/cacheserver`

3. When you are recovering a failed system or want to move (or clone) an operational system, restore the backed up directories exactly as they were on the target system, using the identical directory structure.

Note that RTI does not currently provide its own offline or online backup utilities, nor does it support the use of GemFire packaged utilities. Perform steps 1 and 2 periodically to make sure that the system is recoverable if it fails catastrophically. Alternatively, you can use the original source files and Provisioning Manager client systems to restore the RTI system.

### Safeguarding the RTI configuration

Everything you do to configure a new system is represented in an XML configuration file:

```
<install_dir>/emc_rti_<version>/common/rtish/etc/cluster-config.xml
```

Every node in the system contains the full RTI configuration because this configuration file is copied to each node whenever a change is made. Therefore, you only lose this configuration if there is a catastrophic failure across the entire cluster.

To be able to “replay” a configuration on an existing cluster, for migration to another cluster, or to rebuild a system from scratch, maintain a master script (an RTISH command file) that documents the current state of the configuration in use. All operators who have access to the system must check in changes to this script and any scripts that it references. This approach guarantees protection and reuse.

Your RTISH command file and `cluster-config.xml` are logically equivalent to each other. The command file contains all of the property names and property value assignments that update the `cluster-config.xml` file. You can look at `cluster-config.xml`
config.xml and identify all of the actions that were performed when the commands were run.

**Backing up and restoring reference data**

Reference data is loaded in GemFire into regions, which, in turn, are configured to be persistent by means of a disk-store in each of the members of the grid. RTI provides the capability to back up reference data using of the GemFire command line tool gfsh.

A disk-store backup provides a secondary copy of the in-memory data that can be used in case of a major system failure. Reference data loaded onto the ingest grid must be exported for backup by connecting to the locator on the ingest grid. Similarly, data loaded onto the distribution grid must be exported for backup by connecting to the locator on the distribution grid.

**Backing up reference data**

To back up reference data, follow these steps:

1. Determine which kind of reference data must be backed up.

   Table 2 shows the type of reference data, the platform regions that hold the data, and the corresponding disk-store that must be backed up for each of the grids.

   Some reference data can be recreated from its original sources. It may be easier and less time-consuming to reload such data from scratch. Also, based on the frequency with which the reference data is refreshed on the system and how recent the backup is, the data being restored might be stale.

2. When you have identified the type of reference data to be backed up, follow these steps to create a disk-store backup:

   a. To determine the locator’s host and port combination to connect to the desired grid, use the rtish command below:

   ```
   rti list properties --instances <grid_app_name>@<host>
   ```

   For example:

   ```
   list properties --instances ingestgrid-one@*
   ```

   Some sample output from this command is as follows:

   Properties of `ingestgrid-one@usxxserrae2m1.home` include:
   - iucs.datacast.version: 6.12.2
   - iups.datacast.version: 6.12.2
   - rti.cache-server.port: 20101
   - rti.cache-server.socket-buffer-size: 32765
   - rti.distgrid.pr.bucketss: 5
   - rti.gateway.socket-buffer-size: 32768
   - rti.gemfire.bind-address: 127.0.0.1
   - rti.gemfire.distributed-system-id: 1
   - rti.gemfire.locators: 127.0.0.1[10334]
   - rti.gemfire.name: ingestgrid-one
   - rti.gemfire.remote-locators: 127.0.0.1[10335]
   - rti.gfe.eviction.heap: 50
   - rti.hasher.algorithm: md5
Of the set of configured properties, the one of particular interest is the \texttt{rti.gemfire.locators} property.

b. From the rti-t/tools/gfsh/bin directory, launch the GemFire gfsh utility and connect to the desired grid locator determined in the previous step.

For information on how to setup gfsh for the platform, refer to
Using GemFire shell. For more information about the gfsh backup disk-store command, refer to the GemFire documentation.

c. Connect to locator using the gfsh connect command. For example:

```
gfsh> connect --locator=127.0.0.1[10334]
```

d. Using the gfsh backup disk-store command, create a backup of the persistent region data for the diskstores shown in Table 30.
For example:

gfsh> backup disk-store --dir=/Users/serrae2/RTI/backup
output:gfsh>backup disk-store --dir=/Users/serrae2/RTI/backup
The following disk stores were backed up successfully

<table>
<thead>
<tr>
<th>Member</th>
<th>UUID</th>
<th>Host</th>
</tr>
</thead>
<tbody>
<tr>
<td>ingestgrid</td>
<td>e65d542b-490f-4068-8c4e-</td>
<td>localhost</td>
</tr>
<tr>
<td>one</td>
<td>cbb9bf4cb8b1</td>
<td></td>
</tr>
<tr>
<td>localhost</td>
<td>9b9f009-6336-4257-ba56-</td>
<td></td>
</tr>
<tr>
<td>ingestgrid</td>
<td>0c749032-c868-4be3-b6c6-</td>
<td>localhost</td>
</tr>
<tr>
<td><a href="mailto:locator@usxxserrae2m1.home">locator@usxxserrae2m1.home</a></td>
<td>ceccd61d-a53d-4d8b-b5fd-</td>
<td>localhost</td>
</tr>
<tr>
<td><a href="mailto:ingest-grid-locator@usxxserrae2m1.home">ingest-grid-locator@usxxserrae2m1.home</a></td>
<td>ceccd61d-a53d-4d8b-b5fd-</td>
<td>localhost</td>
</tr>
<tr>
<td>ingestgrid</td>
<td>a47b5da8-dd09-401c-8d07-</td>
<td>localhost</td>
</tr>
<tr>
<td>locator</td>
<td>f7310ef9d093</td>
<td></td>
</tr>
<tr>
<td>ingestgrid</td>
<td>c6125f37e4859</td>
<td>localhost</td>
</tr>
<tr>
<td>locator</td>
<td>ceccd61d-a53d-4d8b-b5fd-</td>
<td>localhost</td>
</tr>
</tbody>
</table>

The target directory specified by the --dir parameter must be available in all members of the grid and can be either local disk or shared storage. A sample backup target directory and its contents can be seen here:

usxxserrae2m1:rdl serrae2$ tree -L 3 /Users/serrae2/RTI/backup
/ /Users/serrae2/RTI/backup
--- 2016-01-06-23-57-31
--- localhost_ingest_grid_locator_usxxserrae2m1_home_38222_locator_v0_59068
--- README.txt
--- config
--- diskstores
--- restore.sh
--- user
--- localhost_ingestgrid_one_38469_v1_19216
--- README.txt
--- config
--- diskstores
Repeat the process for any other grid whose disk-stores you wish to create a backup using the data types listed in the table below.

**Restoring reference data**

To restore a copy of backed up reference data, follow these steps:

1. Locate the directory in which the backup was created and `cd` to it.

   The folder structure contains a top folder based on the date that the backup was created (for example, `2016-10-06-23-57-31`), followed by a directory named after the host and grid application name (for example, `localhost_ingestgrid_one_38469_v1_19216`).

2. Copy the entire contents of this directory to the server location where the backup is being restored.

3. Run the `restore.sh` script located in that directory.

4. Bring the node member of the cluster back into service by executing the required steps in Configuring a multi-host system.

   Depending on the type of failure which is being recovered from, this step may entail the creation of entire new node to join the cluster, or perhaps just a stop and start of the grid member to pick up the new region data that was brought back via the restored backup files.

The table below lists the reference data types and their corresponding disk-store type. Use this information when taking a backup of reference data.

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Region</th>
<th>Grid(s)</th>
<th>Disk-store</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cell towers</td>
<td>REF_CellTowers</td>
<td>Ingest and distribution</td>
<td>refdata</td>
</tr>
<tr>
<td>Devices</td>
<td>REF_MobileEquipment</td>
<td>Ingest and distribution</td>
<td>refdata</td>
</tr>
<tr>
<td>Subscriber profiles</td>
<td>REF_SubscriberProfiles</td>
<td>Distribution</td>
<td>refdata</td>
</tr>
<tr>
<td>Organization units</td>
<td>STREAM_OrganizationUnits</td>
<td>Distribution</td>
<td>refdata</td>
</tr>
<tr>
<td>Geofences</td>
<td>SUBSCRIBERS_GeoFence</td>
<td>Ingest</td>
<td>subscriberdata</td>
</tr>
<tr>
<td>External subscriber opt-ins</td>
<td>OPTIN_EXTERNAL_Subscribers</td>
<td>Ingest</td>
<td>subscriberdata</td>
</tr>
<tr>
<td>External location opt-ins</td>
<td>OPTIN_EXTERNAL_Locations</td>
<td>Ingest</td>
<td>refdata</td>
</tr>
<tr>
<td>Global location opt-ins</td>
<td>OPTINLOBAL_Locations</td>
<td>Ingest</td>
<td>refdata</td>
</tr>
<tr>
<td>Global subscriber opt-ins</td>
<td>OPTINGLOBAL_Subscribers</td>
<td>Ingest</td>
<td>subscriberdata</td>
</tr>
</tbody>
</table>
Other backup requirements

Back up the SSH keys that are in use for RTI user accounts.

Back up any operating system configuration parameters, such as memory and network settings. These are external to RTI, but they must be backed up and recorded for reference.

Recovering from a system failure

This section describes procedures for recovery from RTI system failures, in particular failures that result from hardware issues with one or more nodes in the cluster.

Planning for data recovery

Currently RTI relies on GemFire Disk Stores to provide recovery of stateful reference data that is loaded into RTI. By default, all reference data is persisted to Disk Stores. Additionally, data serialization metadata is persisted by GemFire to a separate Disk Store. This metadata is needed to recover the application reference data.

The underlying GemFire data management technology acts as a distributed database and provides key features that enable continuous availability and dynamic changes to capacity (with no system downtime). To prevent loss of data or system downtime, it is critically important that redundancy be configured higher than "0" (Zero). Most customers keep the default value of "1", meaning that the system maintains One Primary and One Secondary copy of all reference/provisioning data at all times. If redundancy is configured to 0 and a grid node fails, all traffic stops until the failed grid node is restarted (with its original Disk Store files available).

The locations where disk store data is persisted for ingest grid and distribution grid instances are:

- `rti-t/var/data/<Ingest Grid Instance Name>/diskstore:` Stores application reference data such as opt-ins, cell towers, devices, and so on
- `rti-t/var/data/<Ingest Grid Instance Name>/gateway:` Stores data serialization metadata

In any condition where a failed system must be restarted, these directories and any files stored within them must be available for recovery of previously loaded reference data.

**Note:** Recovery from diskstores is far faster than loading from external sources, especially for system deployments that include large amounts of reference data.

If redundancy is set greater than zero using the properties `rti.ingestgrid.pr.redundancy` (for ingest grid instances) and `rti.distgrid.pr.redundancy` (for distribution grid instances), the complete loss of a
single node from the system does not result in any data or system availability loss. If the value of these properties is $n$, you can lose $n$ server instances before losing all copies of at least some data. The following analysis assumes that redundancy has been set to 1, meaning one primary copy of data through which all update operations occur, and one secondary copy. All data updates are transactionally updated to all copies, so there is no risk of failures leading to stale data.

The underlying GemFire data fabric organizes RTI’s subscriber-related data into buckets that assure the grouping of related data (providing the fastest possible processing due to locality). Each logical bucket has one primary copy and $n-1$ secondary copies. These copies are automatically guaranteed to be on different server instances. Contact RTI Technical Support if you plan to deploy more than one RTI server instance per underlying physical host; in this case, there are additional advanced configuration steps needed to ensure that buckets are never allocated to the same physical host (leveraging GemFire’s “redundancy zone” configuration option).

**If $n$ or fewer server instances are lost**

There is no interruption of service. There will be a brief pause in some event processing as the following automated steps occur:

- The surviving cluster members complete logic to verify the status of the lost servers.
- For any primary buckets that were hosted on the lost server(s), the corresponding secondary buckets are immediately promoted to primary status.
- Client applications (ingester, Provisioning Manager, and so on) automatically react by re-establishing lost connections to the remaining servers and updating their internal routing tables for optimized “single-hop” direction of all operations (that is, ensuring that for any subscriber-related event, the event is routed directly to the server hosting the primary bucket for that subscriber).

You can tune how aggressively the system attempts to re-establish redundancy in response to the loss of secondary bucket copies by setting the property `rti.subscriberdata.recovery-delay`. The default value is 600000 milliseconds (10 minutes). This provides a sufficient window of time for an operator to restart a failed server process (or bring a new server online) to re-establish redundancy. This approach minimizes the overhead of rebalancing or moving data between nodes to re-establish redundancy. To apply a more aggressive approach, set the recovery delay as low as zero, meaning immediate recovery of lost redundancy using the remaining available servers. This means that overhead is incurred twice: once for the immediate recovery of bucket redundancy, and once to rebalance buckets later when any failed servers are brought back online.

Be careful if the root cause of the server failure is system overload from too much reference data, too much event traffic, or both. While RTI has multiple levels of protection (such as disk overflow) to prevent overload, any system can be pushed to the point of failure if the volume of traffic is too high. GemFire-based systems that are close to failure because of too much resource consumption might benefit from the loss of redundancy caused by server failures. This is because the overall amount of data replication in the system is reduced when redundancy is lost.

Restarting a failed server when the remaining servers are just below the threshold of resource over-consumption might tip the balance toward a cascading cluster failure. In
such cases, it is usually recommended that operators wait until the system is less busy before adding server nodes back to the system. It may also be advisable to add more than just the failed server nodes to prevent a recurrence of failure due to insufficient resources. To facilitate this, you can create, deploy, and start additional server instances on-the-fly to a live RTI system.

**Important:** Because this behavior depends heavily on system usage discuss a customized plan for handling server crash and system capacity management scenarios with an EMC RTI Systems Engineer.

To restore failed server Instances, take one of these actions:

- **Restart the failed server:** If the hardware is healthy, the best option is to restart the failed server. This restores the original capacity/configuration with the minimum of system overhead. Provision a new host with RTI and enable the failed server instance there (or use a spare that is pre-provisioned for this purpose).

- **As long as there is at least one host where RTI (or an RTISH control/master node) is deployed, run RTISH to deploy the system to the new host and enable application instances on new or existing hosts. When the server instance is deployed or enabled on a new host, start it to restore the original capacity. This assumes the host has similar hardware resource capabilities and is on a local LAN with the other server instances.

- **Force the system to "rebalance":** This allocates new instances of lost buckets and re-establishes redundancy. Use the RTISH command `rti system rebalance` (but first use `rti system connect` to connect to the targeted server cluster to execute this command). Use this option if you are unable to restart the failed server or allocate a new one.

**If more than n servers are lost in a failure event**

In this case, the system loses some buckets entirely, so take quick action to restore them. When this situation occurs, updates that map to the lost buckets fail, and the system logs warnings and exceptions.

First, try to establish the cause of the failure and to determine whether the failed server is in a state where it can be safely restarted (for example, it does not have a corrupted file system or a bad NIC). As with a failure of n or fewer servers, restarting the failed server is the most favorable option. The biggest problem with this kind of failure is that the system does not start if persistent copies of failed buckets are missing.

Two courses of action are possible to remedy this:

- **After creating and deploying a new instance or enabling the same instance on a different host (but before starting it):** Copy the disk store files from the failed instances file system to the same relative path location on the new host file system. When you start the instance, it will use these disk store files to restore the missing data into the system.

- **If it is not possible to restart the instance on the same host, revoke the missing disk store partitions using GFSH.**
## RTI error messages

The following table lists the error messages returned by RTI components other than RDL.

<table>
<thead>
<tr>
<th>Category</th>
<th>Message</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLI</td>
<td>Original exception thrown for <code>%s.%s(%s)</code></td>
<td>Output of a thrown system exception with full stack trace.</td>
</tr>
<tr>
<td>CLI</td>
<td>Cannot modify the configuration. Check your privileges!</td>
<td>Check system privileges for operation.</td>
</tr>
<tr>
<td>CLI</td>
<td>Unable to select the local instance from known hosts</td>
<td>Unknown local instance for configured hosts</td>
</tr>
<tr>
<td>CLI</td>
<td>The '%s' directory on %s does not exist!</td>
<td>Provided directory on host does not exist. Create the missing directory.</td>
</tr>
<tr>
<td>CLI</td>
<td>Error occurred while copying files to %s!</td>
<td>Error occurred during the coping of RTI binaries to host.</td>
</tr>
<tr>
<td>CLI</td>
<td>Cannot remove host rtish is currently running on</td>
<td>You cannot remove the host RTISH executes from.</td>
</tr>
<tr>
<td>CLI</td>
<td>Failed to stop instances on host {}</td>
<td>Unable to halt processes on provided host.</td>
</tr>
<tr>
<td>CLI</td>
<td>Unknown application type</td>
<td>Application type provided is invalid. Refer to the documentation for the supported applications.</td>
</tr>
<tr>
<td>CLI</td>
<td>Instance name can only contain alphanumeric characters or '-'</td>
<td>Provide a valid instance name.</td>
</tr>
<tr>
<td>CLI</td>
<td>No hosts were found</td>
<td>Hosts provided cannot be found. Check configuration.</td>
</tr>
<tr>
<td>CLI</td>
<td>The <code>config.properties</code> file is missing</td>
<td>If you have created the property file using the <code>rti set property</code> command and this error is raised, contact support with this error.</td>
</tr>
<tr>
<td>CLI</td>
<td>Error loading template <code>config.properties</code> file</td>
<td>If you have created the property file using the <code>rti set property</code> command and this error is raised, contact support with this error.</td>
</tr>
<tr>
<td>CLI</td>
<td>Unable to save and distribute config</td>
<td>Failure to write <code>config.properties</code> to host disk or distribute file. Ensure correct permissions are granted.</td>
</tr>
<tr>
<td>Category</td>
<td>Message</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>---------</td>
<td>-------------</td>
</tr>
<tr>
<td>CLI</td>
<td>Exception saving to hostname</td>
<td>Failure to write <code>config.properties</code> to host disk. Ensure correct permissions are granted.</td>
</tr>
<tr>
<td>CLI</td>
<td>Resource loading failed: appType</td>
<td>Loading of the resource while executing <code>rti list properties</code> failed. Refer to the log files for a full stack trace.</td>
</tr>
<tr>
<td>CLI</td>
<td>Error executing shutdown on instance</td>
<td></td>
</tr>
<tr>
<td>CLI</td>
<td>No RTI Application instances found to start</td>
<td>Ensure applications have been configured correctly and deployed.</td>
</tr>
<tr>
<td>CLI</td>
<td>Exception thrown during rebalance operation</td>
<td>Data rebalance operation has failed due to a cluster exception. Contact support with log file details.</td>
</tr>
<tr>
<td>CLI</td>
<td><code>rti.root</code> environment variable is not set</td>
<td><code>rti.root</code> environment variable needs to be set.</td>
</tr>
<tr>
<td>CLI</td>
<td><code>rti.root</code> value of <code>rtiRootDir</code> is invalid</td>
<td>Provided <code>rti.root</code> variable has incorrect format. Example: <code>/var/lib/rti</code></td>
</tr>
<tr>
<td>CLI</td>
<td>Invalid packaging. <code>rti-t</code> directory is missing.</td>
<td>Provided <code>rti.root</code> installation path is missing. Check packaging path is correct and/or amend <code>rti.root</code> variable to point to correct location.</td>
</tr>
<tr>
<td>CLI</td>
<td>Instance name <code>rtiInstanceName</code> invalid</td>
<td>Format of the RTI instance and hostname name is invalid. Format should be <code>instance@hostname</code></td>
</tr>
<tr>
<td>CLI</td>
<td>Hostname <code>hostname@21^z</code> contains invalid characters.</td>
<td>Hostname contains invalid characters. Rename hostname to use alphanumerics.</td>
</tr>
<tr>
<td>CLI</td>
<td>Instance name <code>instance@21^z</code> contains invalid characters.</td>
<td>Instance contains invalid characters. Rename instance to use alphanumerics.</td>
</tr>
<tr>
<td>CLI</td>
<td>Host <code>unknownRTIHost</code> does not exist.</td>
<td>Provided host does not exist. Check configuration is correct.</td>
</tr>
<tr>
<td>CLI</td>
<td>Instance <code>unknownRTIInstance</code> does not exist.</td>
<td>Provided instance does not exist. Check configuration is correct.</td>
</tr>
<tr>
<td>CLI</td>
<td>Not connected! Use <code>rti connect</code> command first.</td>
<td>First perform cluster connection command before issuing commands.</td>
</tr>
<tr>
<td>Category</td>
<td>Message</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>-------------------------------------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>CLI</td>
<td>Instance must be registered with a host</td>
<td>An RTI instance needs to be bound to a host. Amend configuration with the required details.</td>
</tr>
<tr>
<td>JMX/CLI</td>
<td>Unable to load the JMX Management Agent</td>
<td>This message is thrown when an agent cannot be loaded into the target Java virtual machine.</td>
</tr>
<tr>
<td>JMX/CLI</td>
<td>Method foo does not exist in mbean object-name</td>
<td>The provided method to execute upon does not exist for provided object.</td>
</tr>
<tr>
<td>JMX/CLI</td>
<td>Malformed bean info.</td>
<td>Bean information returned is malformed using the <code>rti list mbeans</code> command.</td>
</tr>
<tr>
<td>JMX/CLI</td>
<td>Method expects 3 parameter(s), got 1</td>
<td>Provide the correct number and types of parameters for given object method.</td>
</tr>
<tr>
<td>Rabbit/CLI</td>
<td>Unable to subscribe for messages.</td>
<td>Check broker has been started. Contact support if this error persists.</td>
</tr>
<tr>
<td>Rabbit/CLI</td>
<td>Exception occurred during 'Wait for ESCAPE key to be pressed' process.</td>
<td>Contact support with error log.</td>
</tr>
<tr>
<td>Rabbit/CLI</td>
<td>Exception occurred during 'Stop consuming messages' process.</td>
<td>Contact support with error log.</td>
</tr>
<tr>
<td>Rabbit/CLI</td>
<td>Failed to perform disconnect.</td>
<td>Contact support with error log.</td>
</tr>
<tr>
<td>Syphon</td>
<td>File name must not contain a path. Specify path through <code>--output.dir</code> flag</td>
<td>File name cannot contain a directory path. Provide only the filename.</td>
</tr>
<tr>
<td>Syphon</td>
<td>Invalid directory</td>
<td>Provided directory cannot be found.</td>
</tr>
<tr>
<td>Syphon</td>
<td>Write permission issues</td>
<td>Cannot provide directory due to incorrect permissions.</td>
</tr>
<tr>
<td>Syphon</td>
<td>Command line argument errors. Unknown command.</td>
<td>Invalid command arguments provided.</td>
</tr>
<tr>
<td>Syphon</td>
<td>Incorrect property format for gemfire locator. Fix format too <code>hostname[port]</code> and try again.</td>
<td>Incorrect format for GemFire locator property.</td>
</tr>
<tr>
<td>Syphon</td>
<td>Hostname <code>myhost$hostname</code> contains invalid characters.</td>
<td>Invalid hostname.</td>
</tr>
<tr>
<td>Syphon</td>
<td>Provided locator port number is not a number. Fix and try again.</td>
<td>Malformed locator port. Correct and try again.</td>
</tr>
<tr>
<td>Syphon</td>
<td>Boolean format exceptions in gzip. Fix and try again!</td>
<td></td>
</tr>
<tr>
<td>Syphon</td>
<td>Command line argument errors. Value missing for <code>{command}</code></td>
<td>Missing argument for the requested command.</td>
</tr>
</tbody>
</table>
Chapter 10: Managing the RTI System

RTI log files

RTI uses the LOGBack framework to provide detailed logs for each RTI application instance. These files are useful mostly for debugging RTI issues at the code level. Edit the logback.xml file per RTI application or application instance to set logging properties and levels. See the LOGBack documentation for basic information about the framework, the contents of the file, and supported logging levels TRACE, DEBUG, INFO, WARN, ERROR: [http://logback.qos.ch/manual/index.html](http://logback.qos.ch/manual/index.html)

In a deployed RTI system, a logback.xml file is created for each application type: ingester, ingest grid, and so on. When you enable application instances, a new logback.xml file is created for each instance to provide finer-grain control over the logging behavior. The LOGBack files are stored in the ~/emc_rti_<version>/etc directories for each application. For example, the following file is an ingest grid file.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
    <jmxConfigurator />
    <appender name="R" class="ch.qos.logback.core.rolling.RollingFileAppender">
        <!-- See also http://logback.qos.ch/manual/appenders.html#RollingFileAppender -->
        <append>true</append>
        <File>${rti.logs}/rti-ingestgrid.log</File>
        <encoder>
            <pattern>%d{ISO8601} [%t] %-5p %c - %m%n</pattern>
        </encoder>
        <rollingPolicy class="ch.qos.logback.core.rolling.FixedWindowRollingPolicy">
            <maxIndex>99</maxIndex>
            <FileNamePattern>${rti.logs}/rti-ingestgrid.log.%i</FileNamePattern>
        </rollingPolicy>
        <triggeringPolicy class="ch.qos.logback.core.rolling.SizeBasedTriggeringPolicy">
            <MaxFileSize>10MB</MaxFileSize>
        </triggeringPolicy>
    </appender>
    <logger name="org.springframework" level="INFO"/>
    <logger name="io.pivotal" level="INFO"/>
    <logger name="rti.trace.log" level="TRACE"/>
    <root level="INFO"/>
</configuration>
```

<table>
<thead>
<tr>
<th>Category</th>
<th>Message</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Syphon</td>
<td>Invalid protocol information found. Correct and try again.</td>
<td>Unknown protocol.</td>
</tr>
</tbody>
</table>
You can set the levels for three different types of logging:

- **org.springframework**: separate logging for all Spring code.
- **io.pivotal**: logging for all RTI Java classes with this prefix; advanced RTI users may want to create separate logs for a specific set of classes at a lower level.
- **rti.trace.log**: logging exclusively for channel-based messages that trace events through the system. Enable or disable tracing for specific applications (ingester, ingest grid, and distribution grid) by setting the `rti.ingester.tracer` property. The `rti.trace.log` file logs trace messages only if this property is set to true.

When you enable instances of applications, RTI log files are created for each instance. If you want to set the same logging level for all of the instances of a given application, set the logging level in the `logback.xml` file before enabling instances. For example, you may want all of your ingester application instances to have the same logging level. If you want an instance of a given application type to have a different logging level, enable the instance first, then set its `logback.xml` file accordingly.

Do not change the name or location of the log files (`FileNamePattern`). By default, log files are generated into the `~/emc_rti_<version>/var/log` directory.

For example:

```bash
$ pwd
/Users/rti_user1/dev/emc_rti_<version>/code/build/full-install/emc_rti_<version>/var/log
$ ls
dist-grid-locator ingest-grid-locator ingestgrid-one distgrid-one
   ingester-one  provisioning-one
$ cd ingestgrid-one
$ ls
rti-ingestgrid-gemfire.log
rti-ingestgrid.log
rti-stdout.log
rti-ingestgrid.trace.log
rti-ingestgrid-gemfire-stats.gfs
```

Note that `rti-ingestgrid.log` is one of three log files in this example. This file contains all of the entries requested to be logged by the RTI code. The `rti-stdout.log` file contains entries logged by the JVM but not logged by RTI (for example, unexpected JVM exceptions and standard output errors). The stdout log file is also the default for Java GC logging (if enabled, but no other log file specified). The two-grid and ingester application types create the trace feature output log, even if tracing isn't specifically enabled (in that case the size remains zero).

A GemFire log file is generated for instances of all application types. The embedded GemFire cache servers (for ingest grid and distribution grid instances) or embedded
GemFire clients (all other types) provide their own logging capability to capture information about various cache-related events (start/stop, reads/writes, connections to other processes, and so on). RTI application code uses these caches, but the RTI grid application-specific information is logged in the LOGBack log file, not the GemFire log file.

The GemFire statistics sub-system also creates its own log file with an extension ending in ".gfs" (for GemFire Statistics). Both GemFire and RTI write statistics to this log for offline analysis of system performance. Statistics never contain any specific application data, and are extremely valuable for RTI Support and field engineers to help troubleshoot and generally tune performance related issues.

Regardless of the application instance name, the grid log files are named as follows:

- rti-ingestgrid.log or rti-distgrid.log
- rti-ingestgrid.trace.log or rti-distgrid.trace.log
- rti-ingestgrid-gemfire.log or rti-distgrid-gemfire.log
- rti-ingestgrid-gemfire-stats.gfs or rti-distgrid-gemfire-stats.gfs
- rti-stdout.log

Each instance has the same set of log files in its own log directory.

Ingester applications have the same set of log files following the same naming pattern. The application type is substituted where appropriate ("ingester", "provisioning", "rest-api").

Locator applications have an rti-stdout.log file and a GemFire log file that is created when you run the instance. This file has the naming convention <locatorname>@<host>.log.

This GemFire cache server/client and locator log files are Java Utility Logging (JUL) files. To configure these files, see the GemFire documentation.

The system automatically performs management of log file rollover and eventual aging-out (when a maximum aggregate disk space usage for any given instance is reached). The default maximum aggregate usage is 1GB of disk space of each log file type. The default maximum log file size is 100 MB before rollover.

The default location of the log files that are created for each application instance is defined using the RTI_LOGS environment variable. To change the default location, use the rti set env command.

**Warning:** The RTI_LOGS directory must be mapped to a local disk drive on the host.

The following example shows how to change the default log file locations for the ingester instance named Ingester1 to /home/lowg1/tmp:

```bash
rtish> rti set env --instances Ingester1@sc1f031 --name RTI_LOGS
         --value /home/lowg1/tmp
Set environment variable:
Ingester1@sc1f031
```
'RTI_LOGS' = '/home/lowg1/tmp' was (/home/lowg1/rti21/rti-t/var/log/Ingester1)

When RTI instances are deployed to multiple servers, logs and stats log are also distributed to multiple machines. RTI provides a RTISH command to list or collect logs and stats logs from all deployed and enabled RTI instances. When listing or collecting the logs, options can be used to narrow down the scope of the logs to be listed or collected.

When collecting the logs, the collected logs are copied to a customer-specified directory. The directory is organized in a structure that the first level subdirectories are named after the host names. In each host subdirectory, there are instance directories that contain the logs of the instances on that host. The instance directory names are the instance names.

The following example lists all the logs from all deployed and enabled instances:

```
rtish> rti logs list
Log files for: dist-grid-locator@localhost:
-rw-r--r--  1 liud17  CORP\Domain Users  22605 13 Oct 16:17 rti-stdout.log
......
Log files for: distgrid-one@localhost:
-rw-r--r--  1 liud17  CORP\Domain Users   178 13 Oct 18:53 meta-rti-distgrid-gemfire-01.log
......
Log files for: ingest-grid-locator@localhost:
-rw-r--r--  1 liud17  CORP\Domain Users  198 13 Oct 18:52 meta-rti-ingest-grid-locator@localhost-01.log
......
Log files for: ingest-grid-one@localhost:
-rw-r--r--  1 liud17  CORP\Domain Users  198 13 Oct 18:52 meta-rti-ingest-grid-locator@localhost-01.log
......
Log files for: ingest-one@localhost:
-rw-r--r--  1 liud17  CORP\Domain Users  178 13 Oct 18:54 meta-rti-ingest-gemfire-01.log
......
Log files for: ingestgrid-one@localhost:
-rw-r--r--  1 liud17  CORP\Domain Users  182 13 Oct 18:53 meta-rti-ingestgrid-gemfire-01.log
......
```

The following example collects all logs from all deployed and enabled instances to targetDir/topDir directory:

```
rtish> rti logs collect --targetDir targetDir --name topDir
Logs/statistics are collected to target directory targetDir under top-level directory topDir
```

For details of RTISH logs command, refer to Chapter 16 RTISH Commands
This chapter presents the following topics:

Using the system tools ................................................................. 190
Validation mode for analytics ....................................................... 192
Sample output report .................................................................. 192
Syphon utility syntax .................................................................. 192
Using GemFire shell .................................................................. 196
Using the REST interface to inspect data .................................... 196
Using the HDFS Transport inspection tool .................................. 201
Using RTISH ............................................................................. 202
Reference data loader (RDL) ....................................................... 203
Using the system tools

Use these tools to administer RTI:

- The Syphon utility
- GemFire Shell
- REST interface to inspect data
- HDFS transport inspection tool
- RTISH
- Reference Data Loader (RDL)

Using the Syphon utility

The RTI system includes a Syphon client utility that connects to the ingest grid. This utility is an optional component of an operational RTI system. Use it at any time to capture event output early in the RTI processing flow, after events have been fully anonymized and filtered by the ingest grid, but before they have been sent to the distribution grid.

The utility runs with various command-line parameters that give you control over the content and storage of the output:

- Filtering by one or more supported protocols
- Duration of the syphon command (number of seconds)
- Number of events to capture
- Output file name and/or directory location
- Rollover policy for output files
- Compression of output files
  - GemFire locator for remote syphon commands
  - Reporting on analytics

See Syphon utility syntax

The output files that the utility creates are JSON format text files; however, these files preserve the integrity of the original binary format events that the ingester received. Therefore, you can use netcat or some other program to “replay” the syphoned output directly into the ingester application. See Replaying syphoned events.

Running the Syphon utility

The Syphon utility runs when you specify a syphon command with one or more optional parameters. The RTI system must be operational (ingester, grids, and locators running), but no special setup is required.

If you are running the syphon command from a remote host (a host other than the host where the ingest grid locator is running), specify the --gemfire.locator parameter to connect to the ingest grid.

The utility is installed in the ~/emc_rti_<version>/tools/syphon/bin directory.
Here are some simple examples. For a complete reference to all of the options, see Syphon Utility Syntax.

Syphon 100 events:

```bash
./syphon --count=100
```

Syphon events for a period of 5 minutes (300 seconds):

```bash
./syphon --duration=300
```

Syphon 50000 events specific to three protocols. Name and compress the output file:

```bash
./syphon --count=50000 --protocols=gngi,gb,adr --gzip=true --output.file=gngi_gb_adr_syphon.dat
```

Syphon 50000 events. Roll over the output file (create a new file) for every 10000 events:

```bash
./syphon --count=50000 --rollover.count=10000
```

Five output files will be created in this case.

Syphoned events are stored in JSON format. For example, here is an example of an s1mme protocol event record:

```json
```

Replaying syphoned events

You can stream the results of the syphon command back into the ingester and "replay" the events through the system. For example, run the syphon command on a production system, then stream the output files back into the ingester on a test system. You can use this feature to verify the number of events that are processed, or to experiment with the same source data on an RTI system with a different configuration.

Follow these steps:

1. Using RTISH, set the json.enabled property to true on the ingester instances. For example:
   ```bash
   rlish> rti set property --instances Ingester1@* --name json.enabled --value true
   ```

2. Optionally, set the port for streaming JSON events into the ingester. The default port is 29009. Change the port by setting the jason.tcp.port property on the ingester instances.
Chapter 11: Using the System Tools

3. Stream one or more output files into the ingester by using the Linux netcat command (or a similar program). For example, the following command streams the data in the specified output file through a TCP connection to host 10.110.125.69, using port 29009.

```
$ cat syphon-out-05022014.094249643.dat | nc -v -w 2 10.110.125.69 29009
```

Validation mode for analytics

Syphon can also be used to track which flags have passed through filters. For all flags, the report generated after you perform the validation indicates the value each flag holds and the value it must hold. This information helps you determine why flags do not pass through certain filters.

To enable analytics validation mode, configuration properties must be set. Then you can use the Syphon command options to perform the type of analytics validation you want.

For more information about setting configuration properties for analytics validation mode, refer to Distribution grid properties and for examples, Configuring the grids. For more information about using the analytics validation options for the Syphon command, refer to Syphon Utility Syntax.

Sample output report

Below is a sample output report:

```
{"imsi":"123456789101112","eventTime":"1439369496000","results":{"sms.mo.2g":[{"parameter":"callType","filtered":true,"actualValue":"11","expectedValues":"9","message":"callType filter failed with a value of: 11. Expected values are: 9"}]
```

Syphon utility syntax

This section describes the complete syntax for the syphon command. The utility is installed in the ~/emc_rti_<version>/tools/syphon/bindirectory.

**Synopsis**

This section shows the complete syntax for the syphon command.

```
./syphon
[ --help ]
[ --output.file=<filename> ]
[ --output.dir=<directory> ]
[ --count=<number of events> ]
[ --duration=<number of seconds> ]
[ --rollover.size=<size> ]
[ --rollover.count=<number of events> ]
[ --rollover.time=<number of seconds> ]
[ --gzip=<true | false> ]
```
Note: Although all of the parameters are optional, you must specify at least one option per syphon command. The syntax .syphon by itself prints the help for the command (equivalent to .syphon --help).

The parameters listed in the section that follows can be specified in any order.

Table 32. Syphon utility command-line parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>--help</td>
<td>Displays help text for the command.</td>
</tr>
<tr>
<td>--output.file</td>
<td>Name of an output file where events are syphoned. By default, the file is named: syphon-json-out.MMDDYYYY.HHMMSSLL where LL is the milliseconds part of the timestamp. Do not use quotes to identify the file name.</td>
</tr>
<tr>
<td>--output.dir</td>
<td>Path to an output directory where events are syphoned. The default location is your current working directory. If you do not specify the --output.dir and --output.file arguments, the output is generated in the current directory where the user is running the utility. Do not use quotes to identify the directory path.</td>
</tr>
<tr>
<td>--count</td>
<td>Total number of events to syphon before exiting.</td>
</tr>
<tr>
<td>--duration</td>
<td>Number of seconds to syphon events before exiting.</td>
</tr>
<tr>
<td></td>
<td>If --count and --duration are both specified, the utility honors the condition that is met first. For example, consider this combination of parameters: ./syphon --count=100 --duration=300 In this case, if 100 events are syphoned in less than 5 minutes, the utility stops running. Note: If neither --count nor --duration is specified, the utility runs indefinitely until the user cancels it, either with Ctrl-C or by stopping the KB that the output file reaches before the output rolls over to a new file. If any of the three rollover arguments are specified, the default file name of syphon-out-MMDDYYYY-HHMMSSMS.dat will take effect, starting with the second file, even if an initial file name was specified with the --output.file parameter. Process. This approach is not recommended.</td>
</tr>
<tr>
<td>--rollover.size</td>
<td>Size in</td>
</tr>
<tr>
<td>--rollover.count</td>
<td>Number of events that a single file must contain before the output rolls over to a new file.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Purpose</td>
</tr>
<tr>
<td>------------------------</td>
<td>-------------------------------------------------------------------------</td>
</tr>
<tr>
<td>--rollover.time</td>
<td>Number of seconds that a single file must use to syphon events before the output rolls over to a new file.</td>
</tr>
<tr>
<td>--gzip</td>
<td>Whether the output files must be compressed. Default: false.</td>
</tr>
<tr>
<td>--gemfire.locator</td>
<td>GemFire locator hostname and port. Default: localhost[10334]. This parameter is needed only when the syphon command is run remotely.</td>
</tr>
<tr>
<td>--protocols</td>
<td>Comma-separated list that specifies one or more of the following protocols: adr gb gngi iucs iups json radius s1mme tdr txt For example: --protocols=s1mme,radius By default, events for all protocols are syphoned.</td>
</tr>
</tbody>
</table>

Options for analytics validation mode

<p>| --analytics.validation | After the system has been configured to enable analytics validation mode, to start analytics validation mode, set this option to ‘true’. For example: --analytics.validation=true The behavior of Syphon changes to generate reports on the systems’ analytics instead of its message data. Output messages are not sent to JSON, but are summarized instead with: • imsis • timestamp • Whether the message met the criteria to pass the analytics filters Most of the other properties, such as duration and output file, work the same in this mode. Use this option in combination with the other options for analytics validation: • analytics.imsis • analytics.flags • analytics.before • analytics.after |</p>
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>--analytics.imsis</td>
<td>Specifies specific IMSIs as a comma-separated list for analytics validation.</td>
</tr>
<tr>
<td></td>
<td>For example: --analytics.imsis=99dcd248953e9ec6c9b45646ed34999, 99dcd248953e9ec6c9b45646ed34999</td>
</tr>
<tr>
<td></td>
<td>If this option is not set, all IMSIs that Syphon receives will be processed as output.</td>
</tr>
<tr>
<td>--analytics.flags</td>
<td>Specifies specific flags as a comma-separated list for analytics validation.</td>
</tr>
<tr>
<td></td>
<td>For example: --analytics.flags=sms.mo.2g, pdp.context.3g</td>
</tr>
<tr>
<td></td>
<td>If this option is not set, all analytics data that Syphon receives will be processed as output.</td>
</tr>
<tr>
<td>--analytics.before</td>
<td>Specifies that only messages with a timestamp before the specified time are to be validated.</td>
</tr>
<tr>
<td></td>
<td>For example: --analytics.before=1000</td>
</tr>
<tr>
<td></td>
<td>Timestamp is in milliseconds.</td>
</tr>
<tr>
<td>--analytics.after</td>
<td>Specifies that only messages with a timestamp after the specified time are to be validated.</td>
</tr>
<tr>
<td></td>
<td>--analytics.after=5100</td>
</tr>
<tr>
<td></td>
<td>Timestamp is in milliseconds.</td>
</tr>
</tbody>
</table>
Chapter 11: Using the System Tools

Using GemFire shell

GemFire GFSH provides a single, powerful command-line interface from which you can launch, manage, and monitor GemFire processes, data, and applications. The tool is bundled with RTI under `emc_rti_<version> /tools/gfsh`.

**Note:** Use of GemFire GFSH is intended only for power users to inspect the contents of the GemFire cache directly or for general users with guidance from EMC Support.

### Enabling gfsh

1. Connect to a JMX Manager node using a locator. Do not configure a distribution grid or ingest grid server as a JMX Manager node.

   GFSH requires the JMX Manager feature. JMX Manager nodes are members that manage other GemFire members (as well as themselves). A JMX Manager node can manage all other members in the distributed system.

2. Apply the following properties to the locator instances only:
   - `gemfire.jmx-manager set to true`
   - `gemfire.jmx-manager-start set to true`

   For example:

   ```
   rti set property --instances Locator1IngestGrid@* --name gemfire.jmx-manager --value true
   rti set property --instances Locator1IngestGrid@* --name gemfire.jmx-manager-start --value true
   ```

   Omit the `rti.` prefix from GemFire properties such as `gemfire.jmx-manager-start` for locators. This rule is in contrast to other RTI application types, where the property name pattern is `rti.gemfire.<gemfire-property-name>`.

   The only locator properties mapped to GemFire properties that do start with `rti.` are as follows:

   - `rti.gemfire.locators`
   - `rti.gemfire.remote-locators`
   - `rti.gemfire.bind-address`

### Using the REST interface to inspect data

The REST interface is a client application that you can run from any system. The interface generates a list of URIs that correspond to GemFire regions where RTI data is stored. Using a browser or a client tool, go to these links to discover information about different types of data that is loaded into the RTI grids.

### Configuring the JAVA_HOME variable

Configure the JAVA_HOME environment variable for the RTI instance before you launch the rest-api instance from RTISH.

Run the following command on the machine running your RTI instance:
rti set env --instances <rest-api-instance name> --name JAVA_HOME --value <jdk_dir>

rtish> rti list properties --instances rest1@localhost
Properties of rest1@localhost:
# Context path to start the application server.contextPath: /rti-t gempire.locators: 10.110.125.80[10334]
# Rest1 Application Port
server.port: 8080
# Rest1 Application Instance Name
rti.instance.name: rest1@localhost

See Configuration properties for details about these properties.

3. Enable an instance of the rest-api application on the RTI hosts.

   For example:

   rtish> rti enable instance --instances rest1@*
   Updated instances: rest1 [rest-api] enabled

4. Start the instance:

   rtish> rti start --instances rest1@localhost
   Executed 'start', successfully on: rest1@localhost status: 0
   restapi.app is running with PID 37386

On startup, the REST application generates a list of uniform resource identifiers (URIs), or href links, for the regions associated with the grid. Each Where </jdk_dir> is the JDK installation directory on the machine.

To set up and run a REST client:

1. Using RTISH, create an application of type rest-api. For example:

   rtish> rti create application --type rest-api --name rest1
   Application rest1 has been created

2. Set the rti.gemfire.locators property for the rest-api application. For example:

   rtish> rti set property --instances rest1@localhost
   --name rti.gemfire.locators --value 10.110.125.80[10334]
   Property rti.gemfire.locators set to 10.110.125.80[10334]

   The REST client uses the locator addresses to connect to the grids. See Configuring an RTI system for information about creating locators.

3. Optionally, list the REST client properties (as stored in the /etc/rest-client/application.properties file).

   For example:

   URI identifies the root path to the data for a specified reference data type: cell towers, mobile devices, subscribers, and so on. The URIs follow the same pattern for each data type and region. For example:
Using these paths as the root address to the data you are interested in browsing, drill down by using two key functions:

- **findByKey**: search on specific data by IDs, such as cell tower IDs.
- **findByLimit**: return a limited set of records for the data type, such as only 20 cell towers.

The key value to search on depends on the data type. For cell towers, it is `lac_CellID`. For example:

```json
http://localhost:8080/rti-t/ref-celltower/search/findByKey?key=175_52555
{
  "cellID" : 52555,
  "cellName" : "W04883022", "sitePostCode" : "RM17 5DX",
  "siteEasting" : 561530.0,
  "siteNorthing" : 178180.0,
  "siteLatitude" : 51.4789951493,
  "siteLongitude" : 0.32639147502,
  "centroidEasting" : 561208.0,
  "centroidNorthing" : 178005.0,
  "centroidLatitude" : 51.4789951493,
  "centroidLongitude" : 0.32639147502,
  "radii90" : 400.0,
  "radii80" : 400.0,
  "radii70" : 400.0,
  "cellType" : "NA", "lac" : 175,
  "_links" : {
    "self" : {
      "href" : http://localhost:8080/cellTowers/175_52555
    }
  }
}
```

The following is another cell tower example, with a limit of two cell tower records as the search criterion:

```json
http://localhost:8080/rti-t/ref-celltower/search/findByLimit?limit=2
{
  "_embedded" : {
    "cellTowers" : [
      {
        "cellID" : 17445,
        "cellName" : "G30942",
```
You can run the same kinds of searches against most of the RTI data types, whether the data was loaded as reference data or provisioned in as part of an event subscription. The following table shows the key values for each supported data type:
Table 33. Key values for reference data types

<table>
<thead>
<tr>
<th>Data type</th>
<th>Region</th>
<th>Key</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cell towers</td>
<td>REF_CellTowers</td>
<td>LAC_cellID</td>
</tr>
<tr>
<td>Devices</td>
<td>REF_MobileEquipment</td>
<td>TAC</td>
</tr>
<tr>
<td>Subscriber profiles</td>
<td>REF_SubscriberProfiles</td>
<td>IMSI</td>
</tr>
<tr>
<td>Organization units</td>
<td>STREAM_OrganizationUnits</td>
<td>OrganizationName_OrganizationUnitName</td>
</tr>
<tr>
<td>Geofences</td>
<td>SUBSCRIBERS_GeoFence</td>
<td>OrganizationName.OrganizationUnitName.GeofenceID</td>
</tr>
<tr>
<td>External subscriber opt-ins</td>
<td>OPTIN_EXTERNAL_Subscribers</td>
<td>IMSI</td>
</tr>
<tr>
<td>External location opt-ins</td>
<td>OPTIN_EXTERNAL_Locations</td>
<td>LAC</td>
</tr>
<tr>
<td>Global location opt-ins</td>
<td>OPTIN_GLOBAL_Locations</td>
<td>LAC</td>
</tr>
<tr>
<td>Global subscriber opt-ins</td>
<td>OPTIN_GLOBAL_Subscribers</td>
<td>IMSI</td>
</tr>
<tr>
<td>Blacklist subscriber opt-ins</td>
<td>OPTIN_BLACKLIST_Subscribers</td>
<td>IMSI</td>
</tr>
<tr>
<td>Blacklist location opt-ins</td>
<td>OPTIN_BLACKLIST_Locations</td>
<td>LAC</td>
</tr>
<tr>
<td>Event data records (per protocol)</td>
<td>adr, gb, gngi, iucs, iups, radius, s1mme</td>
<td>Event ID</td>
</tr>
</tbody>
</table>

You can browse the data currently stored in the ingest grid or the distribution grid or both. To browse both grids, start up two separate REST clients. The rest-shell command-line tool is available as an optional download that you can install with the Homebrew package manager. Another means of viewing the information returned by the RTI REST interface is CocoaRest, which is a Mac OS X user interface that you can download.

For more information about REST interfaces, see the Spring Data REST Reference Documentation.
Using the HDFS Transport inspection tool

RTI also ships with a tool for introspecting the Hadoop file system where the RTI HDFS Transport Sub-system is writing the data. The script can be found in the tools directory: tools/hdfs-introspector/bin/hdfs-introspector

When running with no arguments, the script prints all the available commands.

Using this tool, you can do the following:

- Find the overall status of the HDFS directories where RTI is writing data. Use the command line switches --overview.base.path and --hadoop.fsUri, for example:

  --overview.base.path=[base path where HDFS Transport for RTI is writing the data]
  --hadoop.fsUri[default is hdfs://localhost:9000]

  This returns a general overview of the HDFS Transport status. HDFS output for LORS, LANDS, Various Metrics, and ESS streams are introspected and output a summary, including each file path, file size on HDFS, the number of events in each file, and so on.

  The base path provided must match with what is given to the HDFS transport application. The default used for HDFS Transport is /rti-t/data, however, it must be explicitly provided.

  The following are valid examples:

  hdfs-introspector --overview.base.path=hdfs://servername:9000/hdfs/transport

  or:

  hdfs-introspector --overview.base.path=/hdfs/transport
  --hadoop.fsUri=hdfs://servername:9000

- Decode the HEX-encoded files:

  --decode.file=[hdfs file URL to decode]
  --decoded.output.file=[local path for decoded output content]

  For compressed gzip data, HDFS Transport encodes the data in hexadecimal notation when writing to HDFS. These options would decode the HEX data and un-compress to the original textual representation.

  Providing an output file is recommended through the --decoded.output.file argument. Otherwise, it would decode the data to the standard output. This must be avoided except for testing with very small files.

- Copy single files from HDFS to local copy destination.

  --copy.to.local=[hdfs URL to copy from]
  --copy.destination=[local copy destination]
Using RTISH

The RTI shell tool (RTISH) is a command-line interface (CLI) that you can use to install, configure, and manage components in the RTI system. Work with components on the machine where RTISH is installed and on remote hosts.

To start RTISH:

```bash
$ cd <rti_root>
$ ./<common>/rtish/bin/rtish
... rtish>
```

You can also start RTISH and run a script directly from the OS shell. For example:

```bash
% path/to/rtish --cmdfile example.cmd
```

RTISH is interactive and supports tab-completion. The `help` command lists all of the available commands. You can also display help for individual commands.

To become familiar with the RTISH interface, use the following commands:

**Table 34. Basic RTISH commands**

<table>
<thead>
<tr>
<th>Task</th>
<th>RTISH command</th>
</tr>
</thead>
<tbody>
<tr>
<td>Show a list of available commands</td>
<td>rtish&gt; help</td>
</tr>
<tr>
<td>Show help for an RTISH command</td>
<td>rtish&gt; help rti create application</td>
</tr>
<tr>
<td>Show the RTISH version</td>
<td>rtish&gt; version</td>
</tr>
<tr>
<td>Show RTISH properties</td>
<td>rtish&gt; system properties</td>
</tr>
<tr>
<td>Run a script from RTISH</td>
<td>rtish&gt; script --file example.cmd</td>
</tr>
<tr>
<td>Exit from RTISH</td>
<td>rtish&gt; exit</td>
</tr>
</tbody>
</table>

The first time you execute RTISH on a host, the tool detects that the host contains a new RTI installation. Everything you do to configure the new system is represented in an XML configuration file:

```xml
<install_dir>/emc_rti_<version>/common/rtish/etc/cluster-config.xml
```

When you deploy RTI to additional hosts, this file is copied and maintained on those hosts as well. Before creating any applications, check the name of your local host (where you are running RTISH):

```
rtish> rti list hosts
```
Assuming that RTI is freshly installed, the output must be a single line. For example:

192.168.0.100
or
srv001.example.org

When you run RTISH for the first time, it may fail to find a globally visible name for the current host. In this case, the system chooses a name that only makes sense in the local context, such as localhost, and returns a warning message. To make the host visible from different machines, use the `rti rename host` command to assign a globally visible name, such as an IP address or a DNS host name, to the host.

For details about the complete set of RTISH commands, see RTISH commands.

**Reference data loader (RDL)**

Reference data can be loaded into the RTI system using the RTISH utility. RTISH also supports certain operations on loaded reference data as described below. We recommend using the RTISH utility in place of the Reference Data Loader utility.

For more information about the RDL, refer to:

- Chapter 4 Loading Reference Data
- Chapter 15 Referencing RDL Commands
This chapter presents the following topic:

RTI on virtualized environments best practices ..................................................205
RTI on virtualized environments best practices

RTI is well-suited to running on virtualized infrastructure. This chapter summarizes some tuning and usage best practices. While these practices are not technically challenging, some might not be compatible with the virtualized infrastructure management practices that are best for different types of applications. Depending on the strategy and available services of your target infrastructure as a service (IaaS), these practices may require more careful planning and cooperation with your IaaS provider.

Prevent memory ballooning/sharing

As a large-scale, high-performance distributed Java application, RTI requires predictable memory performance at all times. Unexpected pauses in application responsiveness due to memory over-subscription on physical hosts can cause delays in event processing, flow-control back-ups in upstream event publishers, and even cluster instability. RTI best practices are to fully reserve all memory allocated to RTI virtual hosts.

Restrictions on VMware vMotion (or equivalent technologies)

Be aware that vMotion on virtual machines (VMs) hosting RTI Grid Server instances might relocate large amounts of data. This includes both ingest grid and distribution grid processes. When vMotion is in use, all network I/O must be physically duplicated to both the original and the target vMotion physical host. Under high throughput situations, this can cause vMotion to take a long time to complete, and can even cause it to fail. It is important to avoid vMotion activity any time that nodes are very busy, or when nodes are joining or leaving a cluster. This can drive RTI’s own rebalancing activity and thus extra network load. RTI best practices are to only allow vMotion to run during periods of low activity and when processes are not being started or restarted.

RTI uses GemFire Locators for location services. vMotion is known to be incompatible with locators, causing cluster instability or even cluster failure. RTI best practices are not to run vMotion for VMs that run locator instances. If vMotion must be used, the stun phase cannot last longer than 15 seconds.

Dedicated physical hosts for production VM’s

Because RTI is a horizontally scaled and performance-sensitive distributed application, it is not a good candidate for directly sharing workloads with other applications on the same physical hosts. Production deployments are usually sized to require the capacity of multiple physical hosts, and adding more hosts scales deployments. Mixing in additional workloads is unadvisable for RTI’s scalability model. Workload traffic can be shared at the network level as long as sufficient capacity is assured at all times for RTI’s needs.

We recommend that VMs running RTI applications be dedicated to RTI only (ingester instances or Grid instances). Tools such as Syphon or reference data loading can be safely run on VMs with mixed workloads. Consider running applications directly related to RTI on the same VMs if the combination is specifically performance-tested and measured, so that the combined workload is part of the planned scalability model. RTI best practice is to be careful when combining non-RTI workloads with RTI workloads on production.

Non-production uses such as development servers, functional and UAT testing, and so on, where more varied response times are acceptable, may be appropriate candidates for mixed workload environments.
Use of availability zones

For hardware-level redundancy, no two application instances of the same type may share a physical host. Virtualization “Availability Zones” must be matched with RTI server process “Redundancy Zones” (configurable for each defined instance) to assure that the virtualization-level and application-level redundancy implementations are in harmony. Such zones may be used for hardware host, physical rack, or even data center redundancy.

Governance

Some requirements for successful virtualization of RTI deployments may be in conflict with the existing stated goals of cloud infrastructure strategies. Specifically, the drive towards higher consolidation ratios and utilization rates might cause operations teams to resist the requirements of dedicated memory resources, avoidance of mixed workloads with non-RTI applications, and avoidance of vMotion when RTI is busy. Ensure that you address these potential conflicts as part of RTI’s implementation and long-term governance on a virtualized infrastructure service.
Chapter 13 Monitoring and Tuning Event Throughput

This chapter presents the following topics:

Monitoring and tuning event throughput ........................................... 208
General procedure to Tune RTI and the supporting infrastructure .......... 208
Iterative process of scaling and monitoring over time ........................ 210
Understanding horizontal and vertical scaling in RTI ............................ 210
Basic scaling approach to increases in throughput .............................. 210
The impact of event throughput overload on different RTI applications .... 211
Tuning flow control and parallelization .............................................. 215
Tuning the ingester ............................................................................ 221
Tuning the ingest grid ................................................................. 224
Tuning the distribution grid ........................................................... 229
Tuning the number of buckets to rebalance partitions ........................ 233
Stateful analytics and metrics .......................................................... 234
Output stream filtering ................................................................. 234
Setting event threshold .................................................................... 235
Monitoring and tuning event throughput

Any system fails when the load exceeds capacity. For RTI, capacity is exceeded when either the aggregate event throughput through one of the grids or the event throughput in a single application instance is too high. To manage flow control, each of the RTI components accepts inbound event streams, produces outbound event streams, and includes highly optimized message queueing.

The RTI queued message processing architecture protects the system from event loss due to unexpected increases in event throughput rates. To prevent the queues from backing up when processing delays occur due to bursts of events that exceed processing capacity, size your system (both vertically and horizontally) such that under typical operating conditions, throughput bursts only briefly exceed the processing capacity of any given application instance.

For more information about tuning your system to prevent event backups, refer to Preventing initial image surge of events and back-pressure.

Tuning RTI and the supporting infrastructure

To tune RTI and its surrounding infrastructure (hardware and software), follow these general steps:

1. Set up methods and procedures to monitor the performance of your RTI system:
   a. Ingester
   b. Ingest grid
   c. Distribution grid

   For more information, refer to Monitoring the system and Scaling and monitoring over time.

2. Based on the results of your performance monitoring, complete the following steps:
   a. Determine whether you need to vertically or horizontally scale the system as described in Understanding horizontal and vertical scaling in RTI
   b. If necessary, scale the system and perform step 1 before continuing with the next step.
   c. After monitoring your system, review the sections in Event throughput overload on RTI applications to understand how RTI works from a performance perspective.
   d. Tune flow control and parallelization, if required. For more information, refer to...
e. Tuning flow control and parallelization.

f. Tune the ingester, if required. For more information, refer to
Chapter 13: Monitoring and Tuning Event Throughput

- **g.** Tuning the ingester
- **h.** Tune the ingest grid, if required. For more information, refer to Tuning the ingest grid.
- **i.** Tune the distribution grid, if required. For more information, refer to Tuning the distribution grid.
- **j.** Tune the number of buckets, if required. For more information, refer to Tuning the number of buckets.

**Scaling and monitoring over time**

To ensure your system runs optimally over time, implement iterative procedures to do the following:

- Monitor all components for threats of breaching safe resource utilization thresholds
- Deploy additional capacity as increased traffic and changes to usage patterns impact utilization
- Understand the new usage patterns and adjust your monitoring, if necessary

**Understanding horizontal and vertical scaling in RTI**

Tuning RTI to handle queue backups and general changes in throughput rates requires a basic understanding of horizontal scaling and vertical scaling:

- **Horizontal** scaling increases processing capacity by spreading the load over more physical hosts. RTI can easily be horizontally scaled due to the combination of partitioning and dynamic rebalancing provided by GemFire. Horizontal scaling is typically appropriate when capacity must be increased to accommodate new source input streams, or when additional RTI features are enabled that were not used previously (increasing overall processing load for the same source input streams).

- **Vertical** scaling increases processing capacity by concentrating the load onto more powerful physical hosts. This includes adding additional memory, CPU, or networking capacity to existing RTI hosts, replacing existing RTI hosts with more powerful ones, or choosing the hardware that is most appropriate for your first RTI deployment based on initial sizing analysis.

At different times, scaling might apply to the whole system, or only to specific parts of the system.

**Note:** EMC recommends that you complete a sizing analysis and preliminary load tests using a production-like configuration of features as part of your system deployment. Contact your EMC RTI technical representative to help you do a sizing analysis when you first deploy RTI or when you plan system changes to accommodate increased load rates or new features.

**Scaling to increases in throughput**

The basic scaling approach to handling increases in throughput depends on where processing bottlenecks or hot spots develop. When processing hot spots occur...
unexpectedly, they are easily identified by backups in the queues that feed them. If there is sufficient excess capacity available on the system, hot spots may easily be alleviated by tuning the number of partitions and threads available to each partition. Otherwise, you must add capacity either vertically or horizontally.

For more information about tuning the number of partitions and threads available to each partition, refer to Understanding ingest grid performance and tuning and the sections within that describe how to tune these parts. For more information about vertical and horizontal scaling, refer to Understanding horizontal and vertical scaling in RTI.

The goal of an efficiently tuned and sized system is to provide good performance while using the hosts’ available networking, CPU, and memory resources at approximately the same rate. For example, a system whose CPU, NIC, and memory capacity are all being used at an average of 70 percent capacity, with peaks to 90 percent, is considered to be exceptionally well-balanced and tuned.

If one or more of the hardware resources is underutilized compared to the others, then the system has headroom in that area. Conversely, the system may have very little headroom, if for example, memory utilization is at 20 percent, network utilization is at 20 percent, and CPU utilization is at 90 percent capacity.

**Event throughput overload on RTI applications**

This section discusses the impact of event throughput on different RTI applications:

- Ingester
- Ingest grid
- Distribution grid

*Note*: RTI 3.0 adds the capability to run protocol adapters inside the ingest grid and without the need for dedicated ingester client processes. This option is more efficient for use cases that do not require IMSI-specific Trace or Blacklist features enabled in the ingest grid. Most existing RTI customers benefit by using this feature. The key benefits are improved overall system performance and reduced operational complexity. Both of these benefits are due to needing less processes overall and less process-to-process hand-offs in the end-to-end RTI event flow. RTI users that enable this feature can expect a 25 percent to 50 percent improvement in Ingestion performance (or a 25 percent to 50 percent reduction in the amount of hardware needed to handle the same volume of events). This improvement is specific to Ingestion only (the ingester and the ingest grid). The distribution grid performance is unaffected by this new feature.

For use-cases that require the IMSI-based Trace or Blacklist features enabled on the ingest grid, use of ingester client-based protocol adapters is still the recommended approach. This is because these features benefit from the co-location of events with their (partitioned) target reference data, and partition-aware routing of events is best accomplished using the ingester-IngestGrid client/server connectivity and routing layer.

The ingester application performs basic and stateless protocol transformation that is relatively lightweight and can potentially process event throughput up to the limit of the host’s network interface. The challenges to scaling ingester throughput depend on the capabilities of the upstream publishers feeding the ingester protocol adapters.
Chapter 13: Monitoring and Tuning Event Throughput

**Monitoring ingester performance**

Publishers that are able to scale their throughput by initiating more parallel connections provide the best ingester performance. This is because the most efficient processing model (and the one implemented by downstream RTI components) is to have the threads that process data directly from inbound TCP sockets perform all the application's work, up to the queuing of outbound events for micro-batched delivery to the next downstream process. This model provides the maximum benefits of programming logic in-lining and optimizes Java memory management and clean-up for the many short-lived objects associated with event processing.

Publishers that scale their throughput by increasing the event rate through a single (or limited number of) TCP sockets require that the ingester use an input queue and input queue processing thread pool, significantly increasing the Java memory management and garbage collection overhead.

There are two reasons for this:

- Any additional time that events spend in queues is time that they can be promoted through the Java heap's memory spaces (generations), adding buffer copy overhead and increasing the risk that objects are promoted to tenured space where collection is much more expensive.
- The introduction of a queue between the input socket processing thread and the business logic processing thread prevents the most efficient use of the Java Thread Local Allocation (TLA) stack.

In any case, if an ingester instance reaches the maximum rate possible after tuning the various options, it can be scaled in two ways:

- Vertically, by using either of the following methods:
  - Relocating the ingester instance to a more powerful host (or adding more processing power to the existing host) and increasing the parallel processing with more connections or threads.
  - Upgrading to hardware with faster CPUs that can provide improved throughput by increasing the event rate that a single thread can handle.
- Horizontally, by using the following method:
  - Spreading the configured adapters over more ingester instances deployed to additional hosts. It is easier to do this when multiple adapters are already deployed to a single instance.

To horizontally scale an ingester instance with a single adapter, deploy the adapter to at least one more ingester instance on another physical host. The upstream publisher must be able to split the flow to target multiple ingester instances (whose service is listening on multiple host-port pairs).

**Understanding ingest grid performance and tuning**

The ingest grid performs the core functions of enrichment and filtering. Like all RTI application types, the ingest grid can scale independently of the upstream ingesters or the downstream distribution grid.

Depending on the number of enabled features, the ingest grid’s workload may be more processor and memory intensive than the work performed by the upstream ingester.
applications. If trace features are not extensively used, then the hardware required for the ingester and ingest grid are often about the same, and may even be less if the upstream ingesters require the use of parsing thread pools and queues. Events received from ingester instances are processed through a synchronous pipeline and then (if not filtered) forwarded onto GemFire WAN Gateway sender queues for delivery to the distribution grid.

**Monitoring ingest grid behavior**

Monitor the following behavior in the ingest grid:

- Under normal operating conditions, no peer-to-peer (p2p) data replication is necessary for event processing. RTI leverages the GemFire single hop feature to ensure that each event from the ingester goes to the ingest grid server with primary processing responsibility for the event's subscriber IMSI. This process enables a streamlined and scalable swimming lane style of parallel processing. If p2p network traffic is observed, it is typically a sign of unstable connections with the upstream ingester instances, which might be caused by system overload, timeouts due to Java garbage collection, or faulty networking hardware. The GemFire statistics that track p2p network traffic are under the DistributionStats category. (Look for processedMessages and receivedMessages.)

- To protect the ingest grid from running out of memory during unexpected and sustained throughput bursts, RTI relies on the WAN Gateway Sender queuing overflow-to-disk capability. If you notice any unexpected lag times in end-to-end event processing, check the status of the WAN Gateway Senders by using JMX or analyzing the GemFire statistics files with the VSD tool. If the WAN Gateway Sender queue backs up for any appreciable length of time, this indicates an incorrectly sized or tuned system.

To optimize throughput in the Gateway Senders, tune the following parameters:

- Concurrent queue or processing thread pool size
- Maximum batch size per queue
- Batch timeout per queue

**Understanding distribution grid performance and tuning**

The distribution grid processes events received from the upstream ingest grid instances synchronously through to the outbound queues (for output to .csv files or AMQP subscribers). Because the target file system and Rabbit MQ server are local to the distribution grid network and are efficient event sinks, the system does not expect back pressure on distribution grid output. Consequently, there is no disk overflow of these outbound event queues. They are used only to gain the efficiency of micro-batching.

Processing of events from the ingest grid is automatically partitioned across all distribution grid instances. Like the ingest grid, the distribution grid flow is optimized to avoid peer-to-peer network traffic between distribution grid instances. Both the file output batchers and the AMQP output batchers have batch size and batch timeout tuning options, though the defaults are usually sufficient to handle high throughput.

**Monitoring distribution grid behavior**

This section describes some of the main performance problems in the distribution grid:

- Insufficient capacity, which causes event queues to back up
Chapter 13: Monitoring and Tuning Event Throughput

- Addition of the Metrics Module requires more resources
- Each ESS requires more routing checks for each event

**Event queue back up**

Insufficient capacity to handle event throughput in the distribution grid causes event queues on the ingest grid WAN Gateway Senders to start backing up. The causes of such a backup can be slow disk drives, network problems for traffic to the RabbitMQ server, and insufficient capacity for the overall event load and the number of enabled features.

**Metrics module addition**

With the addition of the metrics module, peer-to-peer traffic may occur between distribution grid nodes. The calculation of aggregate functions for entities other than IMSI (in other words, cell towers, devices, or subscriber groups) requires the use of alternate partitioning and GemFire’s distributed function execution service to perform map-reduce processing. Before implementing metrics, test the impact of enabling them on systems previously sized to use only other RTI features.

**ESS impact on the system**

The load on the system is driven by the combination of inbound event throughput and the degree of outbound multiplexing to subscribing applications. Each ESS increases the number of routing checks needed to process each event. This means that the load is linearly increased for each ESS, even when it is only interested in a small subset of the overall event stream. Load is also increased in direct proportion to the average number of subscriptions that intersect each event.

When you must make changes to the number of subscriptions and the number of their associated opt-ins, ensure you institute procedures to carefully measure and monitor their impact on CPU utilization, network utilization, and overall event processing latency. You can determine the impact of increased resource use for your specific deployment configuration and event patterns. Ensure that .csv output files are processed and deleted in a timely manner; high throughput deployments fill disk space quickly.
Chapter 13: Monitoring and Tuning Event Throughput

Tuning flow control and parallelization

A combination of queueing and network traffic buffering (socket buffers) provide flow control through the system. Parallelization, or parallel processing, is achieved by splitting single-threaded streams into multiple sub-streams, each handled by their own thread. Parallel processing always occurs in the transition between system components, where one set of worker threads feeds one or more queues. Another set of worker threads takes the queued events and either performs the next piece of processing, or sends them to the next downstream process for the next piece of processing for example, from ingest grid to ingest grid, or from ingest grid to Distribution/Service Grid).

Each application type accepts inbound message traffic in parallel over multiple network sockets and, in turn, publishes events to the next component in the overall system. The outbound traffic is always queued before being transmitted over the network for micro-batching and flow control tuning parameters are available to optimize the sizing of micro-batches and worker thread pools to match the capabilities of the underlying hardware infrastructure.

Additionally, the ingester provides the option of an input queue (Parsing Thread Queue) that directly handles the event stream from upstream publishers. The Parsing Thread Pool parallelizes the processing of events from this queue.

To optimize flow control and parallelization, consider whether you need to optimize any of the following components:

The following queue management parameters as described in Tuning queue management:

- Concurrent queues and handlers
- Maximum batch size
- Batch timeout

- The following ingester input event parsing thread pool and queue as described in Tuning the ingester input event parsing thread pool and queue:
  - `<protocol name>.parsingThreads`
  - `<protocol name>.parsingQueueLength`

- The following areas of network socket management and associated receiving end processor threads as described in Tuning network socket management and associated receiving end processor threads:
  - I/O Memory Assigned to Socket Buffers
  - Java I/O Threads Allocated to Parallel Processing

- Java Memory and garbage collection (GC) as described in Tuning Java memory and garbage collection:
  - Reference data caching
  - Flow control queues
Chapter 13: Monitoring and Tuning Event Throughput

- Metrics and analytics
- Stateful enrichment
- Basic tuning recommendations for all RTI applications

**Tuning queue management**

Use the following parameters to optimize queue management under these categories:

- Concurrent queues and handlers
- Maximum batch size
- Batch timeout

Processing batches of events is triggered by one of the following, conditions whichever comes first:

- The maximum batch size being reached
- Batch timeout timespan passing since the last batch was processed

**Concurrent queues and handlers**

Each logical queue is internally implemented as multiple parallel and concurrent queues, with the sole exception of the ingester input adapter parsing threads queue (because this directly absorbs the upstream publisher's event stream). The following properties control the degree of parallelism and can scale with the number of available processing cores:

- **Ingester**: `rti.ingester.<protocol name>.batch.poolsize`, where protocol name is a valid standard or custom-build protocol adapter name (such as adr, gb, iucs, iups, s1mme, gngi, radius, json, and so on).
- **Ingest grid**: `rti.ingestgrid.gateway-sender.dispatcher-threads`
- **Distribution grid**: The distribution grid provides fine-grained control for the LORS and LANDS flows, and for file or AMQP output targets.

**Maximum batch size**

Maximum batch size defines the maximum number of events to accumulate in a queue before processing them as a batch. A larger batch size typically trades an increase in overall throughput for an increase in the average per-event processing latency.

- **Ingester**: `rti.ingester.<protocol name>.batch.size`, where protocol name is a valid standard or custom-build protocol adapter name ("adr", "gb", "iucs", "iups", "s1mme", "gngi", "radius", "json", and so on).
- **Ingest grid**: `rti.ingestgrid.gateway-sender.batch-size`
- **Distribution grid**: The distribution grid provides fine-grained control for the LORS and LANDS flows, and for the file, AMQP, or custom format output targets. The parameter to configure batch size is: `rti.distgrid.<lors|lands>.<file|amqp|custom-format-name>.batchSize`
- Distribution grid: Use rti.subscriber.events.batch-size for output to .csv files, and rti.distgrid.amqp-batch.num-events for output to external/AMQP event subscriptions.

**Batch timeout**

Batch timeout defines the maximum amount of time to wait before processing whatever number of events has accumulated as a batch. When the system is running at full speed, the maximum batch size is usually reached before the batch timeout. When event throughput slows down, batch timeout places an upper limit on message processing latency.

- **Ingestor:** rti.ingester.<protocol name>.batch.timeout (measured in milliseconds), where protocol name is a valid standard or custom-built protocol adapter name (such as adr, gb, iucs, iups, s1mme, gngi, radius, json, and so on).

- **Ingest grid:** rti.ingestgrid.gateway-sender.batch-time-interval (measured in milliseconds)

- Distribution grid: The distribution grid provides fine-grained control over the LORS and LANDS event output endpoints. File, AMQP, or custom format output targets can be fine-tuned to received events in batches to improve event throughput. The parameter you use to configure batch timeout is:
  rti.distgrid.<lors|lands>.<file|amqp|custom-format-name>.timeout

- Distribution grid: Use rti.subscriber.events.batch-time-seconds for output to .csv files, and rti.distgrid.amqp-batch.num-seconds for output to external or AMQP event subscriptions (both measured in seconds).

**Tuning the ingester input event parsing thread pool and queue**

Starting with RTI 2.1, a new thread pool and associated concurrent read and write flow-control queue has been added to parallelize the input parsing performed by protocol adapter codecs. This allows you to vertically scale:

- Input event streams that cannot be spread across multiple ingester instances
- Multiple TCP connections for the same ingester instance (due to limitations of the upstream publisher)

The following tuning options are available:

- `<protocol name>.parsingThreads` -- Controls the number of threads taking events from the input event queue, parsing them (in parallel), applying any enabled filters, and placing them onto the output queue for further processing on the ingest grid.

  Setting this property to a value greater than 1 enables parallel processing of input event parsing. Otherwise, a single thread handles the events directly from each TCP socket.

- `<protocol name>.parsingQueueLength` -- Controls the input event queue size, providing a buffer for enhanced flow control with the upstream event publisher.

  In general, configure the smallest possible parsing queue length without negatively impacting the upstream event publisher. The longer events stay in the queue, the
greater the Java garbage collection overhead required to clean up the event objects after they have been processed.

This section describes the following aspects of how to tune network sockets and the associated receiving end processor threads:

- I/O memory assigned to socket buffers
- Java I/O threads allocated to parallel processing

**I/O memory assigned to socket buffers**

The Receive/Send Socket Buffer Size parameter controls the amount of native I/O memory assigned to socket buffers to help manage flow through TCP/IP. Because most of RTI is based on micro-batching, relatively large amounts of data are sent between application instances (compared to systems that process one event at a time).

To optimize network throughput between layers of the system, calculate the expected maximum size (in bytes) of each micro-batch, and assign this value as the socket buffer size for both the sending and downstream receiving sockets.

Note that many companies deploy Linux with default maximum socket buffer sizes considerably smaller than what is optimal for RTI. If messages in any RTI process log file indicate that Java was unable to allocate the requested input or output buffer size for a socket, ask your Linux administrator to increase the maximum size on all your RTI-deployed hosts.

**Java I/O threads allocated to parallel processing**

Ingest grid instances act as GemFire cache servers and accept connections from GemFire client pools embedded in the ingester client instances. The ingest grid property `rti.cache-server.max-threads` controls the number of Java NIO threads allocated to process event micro-batches from ingester application instances in parallel.

To ensure that no upstream event publisher threads must wait for a server-side thread to become available, set `MaxThreads` at least as high as the aggregate number of `rti.ingester.<protocol name>.batch.poolsize` pool threads working across all deployed ingester application instances.

**Tuning Java memory and garbage collection**

RTI applications mainly handle short-lived objects related to the processing of in-flight events. Therefore, memory and garbage collection (GC) tuning is mainly tailored towards assuring that temporary objects related to event processing are cleaned-up in the younger generations, before being promoted and becoming far more expensive to clean up. Most Java objects are short-lived, even in RTI.

These are the areas where longer-lived objects may exist in RTI:

- Reference data caching
- Flow control queues
- Metrics and analytics
- Stateful enrichment
• Basic tuning recommendations for all RTI applications

**Reference data caching**

Even in the largest deployments, reference data caching does not add up to more than a few gigabytes of memory. All reference data loaded by RDL, RTISH, or the Provisioning Manager resides in tenured space. Typically, these memory-cached data sets do not experience a high amount of churn as they are updated on a daily or hourly basis. Therefore, their impact on GC is low.

If you expect your reference data set to grow, consider increasing the heap size by the same amount. If you expect to increase the update rate of reference data sets, consider increasing the old space size (relative to the new space size) to provide additional GC space.

**Flow control queues**

When flow control queues fill or back up for any reason, the objects they contain may survive multiple GC cycles and be promoted to tenured space. Minimize the chances of this happening by configuring a generous new generation size.

**Metrics and analytics**

The metrics and analytics modules consume memory to cache the contents of tumbling windows, aggregate values, and so on. Objects related to metrics and analytics may also have shorter lifecycles and contribute significantly to tenured space memory churn. The impact is highly dependent on the specific features used, their configuration, and the event patterns they handle.

For help in analyzing the GC impact of your specific metrics and analytics usage, consult with your RTI technical representative.

**Stateful enrichment**

Various enrichers are available in the RTI product to enrich events with values previously captured from other events. Stateful enrichment requires additional memory resources to handle the caching of enrichment objects and potentially the addition of related and added objects.

Using the enricher features increases the memory needed for object caching and the headroom needed for object clean-up.

**Basic tuning recommendations for all RTI applications**

The basic recommendations for all RTI applications are as follows:

- *Use CMS/ParNewGC GC algorithms.* These GC algorithms provide the best combination of performance and stability under various conditions. EMC does not recommend using the Java default throughput collector because of potential process stability issues under unexpectedly high system load. A potential throughput improvement is in the range of 10 percent to 15 percent.

- *Configure a large New Space.* Most of the new space churn in a busy RTI system is due to short-lived event objects. The goal is for objects related to event processing to die as early as possible in the sequence of generational Java GC promotion.
To accomplish this, configure a large new space, helping to assure that spikes in traffic volume do not fill up the new or survivor spaces, which then causes premature promotion to tenured space, or even direct-to-tenured object allocation. New space size can also effect the time that events spend in queues by assuring that downstream system components or I/O are never blocked. The new space size must be at least half of the total heap size. You might find through testing that the optimal size is as high as 2/3 of the total heap size.

- **Configure a generous Thread Local Allocation (TLA) Stack Size.** RTI has been developed using inline processing logic whenever possible. This enables Java to use the TLA stack for allocating most of the short-lived objects related to event processing. However, because processing threads typically take micro-batches of events from inbound socket connections, this significantly multiplies the number of objects that can potentially benefit from the stack size. Configure at least 4 MB for the TLA stack size ("-Xss" tuning switch).

- **GemFire Resource Manager and CMSInitiatingOccupancyRatio:** The RTI GemFire Resource Manager is configured to block all updates when memory utilization reaches 90 percent. All flows between RTI applications include the use of GemFire regions as event pathways, so this also blocks event flow that doesn't otherwise trigger cache updates. This helps to protect RTI from Java OutOfMemory Exceptions, which can corrupt processes.

To ensure that Java GC starts up and attempts to free up memory before this critical threshold is reached, we recommend that you configure the CMSInitiatingOccupancyRatio no higher than 80 (triggering GC clean-up at 80 percent of the heap).

Always include the flag -XX:+UseCMSInitiatingOccupancyOnly to assure that the CMS collector respects your configured initiating occupancy ratio. Otherwise, the CMS collector considers the initiating occupancy ratio as a guideline rather than a firm directive. For RTI, it is important that CMS collection ALWAYS starts up before risking a breach of the Resource Manager’s configured 90 percent critical threshold.
Tuning the ingester

Figure 18 shows an ingester instance with three separate adapters configured and enabled. External event publishers stream events to each adapter by one or more socket connections.

**Note:** Each adapter listens with a server socket on a single port. For each connection request from a publisher, a new thread is spawned that is dedicated to reading the events from the socket. RTI does not control how many parallel socket connections there are at runtime because all connections are initiated from upstream (and all are accepted). You must understand the behavior of each upstream event source publisher to optimally configure the available ingester thread pool and queueing tuning options.

Figure 18. Sample ingest Grid

On the far right is Example ingest grid 1, which is the simplest grid showing only a single inbound TCP connection and no parsing thread pool/queue.

Example ingest grid 2, in the middle, is similar, except that the event stream publisher has dynamically initiated two socket connections so that it can publish dual, or parallel, streams (typically so that it can achieve a higher aggregate throughput rate).

In both of these cases, the Netty Adapter socket listener threads, which take events from their respective sockets, parse each new event using the Adapter's codec logic, apply any
enabled filters, and then queue each event for batched delivery to the ingest grid by way of the adapter's dedicated outbound reactor queue.

Example ingest grid 3, on the left, shows an adapter with the parsingthreads property set to "3", which enables input event stream parsing work flow-control and parallelization. The three threads are represented by the vertical arrows with a "T" stamp on them. This configuration allows event stream publishers to publish at a much higher rate over each socket connection that they open, as the work of parsing and applying filters is performed in parallel using multiple CPU processing cores. Without such parallelization, an event stream publisher is limited in the event rate they can publish by way of a single socket connection.

The choice of whether to configure an adapter with a parsing thread pool depends on the limitations of the upstream publisher. Ideally, the preference is that RTI publishers scale throughput by adding more parallel connections rather than streaming at a higher rate over the same connection, because this allows the in-lining of most event processing work. This is how RTI efficiently scales process-to-process parallel *swimming lanes* processing. Due to in-lining efficiencies in Java, it is always more efficient to initiate or continue parallel processing from upstream, rather than introduce input queues that feed executor thread pools.

**Note:** In-lining is when a single thread completes an entire unit of work in a single stretch, without handing off to other threads. In-lining of logic processing enables Java to allocate memory more efficiently, and subsequently to avoid most Java garbage collection overhead.

---

**Tuning the parsing thread pool and queue**

To optimize your ingester input processing, control the parsing thread pool size and the parsing thread queue size. Consider these trade-offs:

- `<protocol name>.parsingThreads` — The larger the pool size, the more processing power is applied towards parsing and filter-checking the inbound events, enabling higher event throughput from the single-source socket connection.

  Never set the value for this option higher than the available number of processing cores because that would provide less throughput due to thread context switching overhead. Additionally, to pull batches of events and transmit them to the ingest grid, some CPU processing resources must be available for the outbound thread pool.

  Ensure that the total number of parsing threads (across all adapters) does not exceed 2/3 of the total processing cores on the host. When calculating the total number of working parsing threads, an adapter without the parsing threads property configured is considered to have one parsing thread (the same one that reads events from the socket).

- `<protocol name>.parsingQueueLength` — This queue acts as the flow-control buffer for the input event stream. Some event publishers (sometimes called probes) are quick to raise alerts or drop events when they sense back-pressure. Increasing the parsing thread queue size helps to alleviate this problem. Ideally, both sides (RTI and the event source) must have enough of a flow control buffer so the load is balanced between them.

  Configure the queue length to be as small as possible while still providing a reasonable buffer to absorb bursts of traffic, but without creating excessive back-
pressure on the source event stream publisher. The smaller the configured size in the RTI ingester, the lower the potential Java GC overhead due to events sitting in the queue for multiple GC cycles. Each cycle that an event survives while still in the queue causes buffer copy overhead between the Java GC generations. If events are promoted all the way to old space, the final clean-up cost is also considerably higher than in the younger event generations.

Initially, leave the parsing thread pool option disabled, and only use it if system testing shows that upstream publishers can only achieve the target throughput by sending a very high rate over each connection socket. For RTI, it is better to scale the throughput rate into a single adapter by creating more parallel connections to that adapter.

To discover the maximum per-socket throughput rate in your environment, perform testing together with the team responsible for the upstream publishers, and then add more socket connections, each streaming up to that maximum rate.

Use the parsing thread pool only when the more efficient and recommended approach is not possible due to upstream publisher limitations.

### Preventing initial image surge of events and back-pressure

Some event publisher technologies that feed RTI ingesters keep a large batch of events in their queues and send those events all at once when they first connect. This can quickly overload the relevant adapter's input buffers or queues. This complicates ingester tuning because configuring a very large input queue can have negative Java GC implications and may also negatively impact the publisher as it experiences back-pressure immediately upon connecting.

Ideally, discuss this with the team responsible for the publisher application and request that normal event stream publishing start without first publishing a batch of tenured events. If this is not possible, use these options to drop the initial image event surge:

- `<protocol name>.dropInitialImage` — Valid values are true or false. When set to "true", drops the first N seconds of events as configured by:
  - `<protocol name>.dropInitialImageTime` — An integer specifying the number of seconds to drop events at startup before commencing normal event processing.

For more information about these properties, refer to Protocol adapter properties.

### Tuning the reactor queue batch size and thread pool size

Events are added to the outbound reactor queue by either the Netty adapter socket listener thread, or, if enabled, the parsing thread pool threads. A separate thread pool is dedicated to each adapter's output reactor queue to parallelize the transmission of event micro-batchers to the ingest grid.

To tune the reactor queue batch size and thread pool size, use these options:

- `rti.ingester.<protocol name>.batch.size` — Events that are smaller in size typically benefit from a larger batch size, while larger events may show optimal throughput with a reduced batch size.

Values between 1000 and 2000 are typically ideal. A lower value might reduce the overall throughput (by increasing the number of network round-trips to process a given number of events), while a higher value ill tends to increase the overall per-event latency.
**Monitoring an Ingest and Tuning Event Throughput**

- `rti.ingester.<protocol name>.batch.threads` — For each batch delivered to the ingest grid, a thread from this pool waits synchronously until all events in the batch have been received, processed, and queued for transmission to the downstream service/distribution grid. That means the batches may spend significant time in an I/O wait state. Because they process so many events at a time, and the target ingest grid node in-lines all event processing logic, the throughput each thread can achieve can be considerable.

**Tuning the GemFire client connection pool size**

Tuning the GemFire client connection pool size typically is not necessary because, by default, the GemFire pool dynamically adds a connection as needed. If you expect a high number of threads to access the pool, consider increasing the MinPoolSize value (the default is four) to avoid the overhead of opening new connections as the system gets busy after lulls in traffic. (The pool also closes connections after a few seconds of sitting idle, forcing new ones to be opened when traffic increases.)

**Java memory and garbage collection (GC) tuning**

The tuning considerations with the largest impact for ingester processes are related to the input parsing queue described in Tuning the ingester input event parsing thread pool and queue. More generally, ingesters don't require any long-term memory storage of reference data, and the benefits of increased memory are only that they provide additional headroom for generational GC algorithms to run efficiently.

EMC recommends a heap size from 12 GB to 32 GB (depending on whether a parsing queue is used and how large it may grow). Use a larger heap size (in combination with a larger parsing queue length) if you want to minimize the chances that upstream event publishers will experience back-pressure. As with all RTI processes, use the CMS garbage collector, and to include the `-XX:+UseCMSInitiatingOccupancyOnly` flag in your RTI_OPTS Java System Properties.

**Tuning the ingest grid**

Figure 19 shows an ingest grid instance connected to two downstream distribution grid instances. The GemFire CacheServer handles inbound traffic. It is part of the GemFire client/server connectivity model that provides:

- Balanced connections across a cluster
- Dynamic scalability
- Automated failover and reconnect logic
- Single-hop routing based on each event’s primaryID value.

**Note:** Processing and reference data are partitioned based on primaryID values (typically IMSI’s). GemFire manages the allocation of partitions to available grid servers, and the routing of events from ingester clients directly to the ingest grid nodes hosting their designated partitions.
Chapter 13: Monitoring and Tuning Event Throughput

Figure 19. Sample ingest grid connected to two downstream distribution grid instances

The GemFire server threads take micro-batches of events from client TCP socket connections and execute all configured ingest grid business logic. This is a highly efficient processing model from a Java GC perspective. The JIT compiler's in-lining capabilities benefit performance, and most memory allocations are kept on the thread local stack as young generations per Java GC best practices.

After codec deserialization, filtering, and enrichment (as well as any trace logic) are executed by a server thread, only the output event destined for the distribution grid is queued for downstream delivery. The WAN single-hop router then handles the micro-batching and transmission of events to the downstream Service and distribution grid server instances. The single-hop router is composed of a set of routes, one for each target distribution grid server. Each route is composed of a GemFire gateway and its constituent set of gateway queues, worker threads, and client pool with connections to only the lone target distribution grid server.

**Tuning the GemFire server**

The GemFire Server component provides TCP connectivity and processing threads to service ingester client requests. The following aspects of the sockets and the threads can be tuned:

- Number of worker threads
- Socket buffer size

**Number of worker threads**

The `rti.cache-server.max-threads` option (default 64) defines the number of worker threads. Ideally, each connection is serviced by a dedicated server thread,
providing the best possible throughput and response time as requests never have to wait for a busy thread servicing some other connection to become available. Since the ingest grid processing is optimized to avoid I/O wait time, these processing threads are typically able to use an entire CPU core while handling a micro-batch of events.

If you deploy your RTI ingest grid instances to servers with more than 64 logical processing cores, or if you notice that the ingest grid is not as busy as you expect it to be, consider raising this value to increase overall CPU utilization. For smaller hosts with less than 32 logical processing cores that are running near full CPU utilization you can reduce context switching overhead and increase the overall throughput by reducing the value from the default of 64.

**Socket buffer size**

The `rti.cache-server.socket-buffer-size` option's default value is 1,048,576 bytes. To optimize this tuning parameter, calculate the expected average micro-batch size being sent by ingester clients and divide that by the number of ingest grid instances. This works because micro-batches are split by the partition-aware single-hop routing to target the servers hosting their assigned partitions, and these partitions are evenly split across the available servers.

For example, if the average event size is 1300 bytes, and the configured batch size on the ingester client is 2000, then the average total micro-batch size in bytes is 2.6 MB. If there are three ingest grid instances, then most batches split into three parts will be a little less than 1MB, and they can be handled in a single socket buffer allocation by each server. The value for this tuning parameter should be configured the same as for the sending socket buffer size on the ingester clients (controlled by the `rti.ingester.gemfire.pool.socketBufferSize` parameter).

**Queue batch size**

The WAN single-hop router controls the queuing and transmission of events to the downstream distribution/service grid. Under many circumstances, the default values do not need to be changed as they automatically scale as more distribution grid nodes are added.

**Socket buffer size**

Configure the `rti.gateway.socket-buffer-size` parameter (the default value is 4,194,304 bytes) to be large enough to handle an entire single batch. The value for this tuning parameter must be configured to match the receiving socket buffer size on the distribution grid nodes (using the same property name).

Increased Java GC overhead is also a negative impact to increasing the batch size. The longer events remain in the queues, the more time they have to be promoted through the Java heap’s generations (making them more computationally expensive to clean-up).
Queue batch timeout

The `rti.ingestgrid.gateway-sender.batch-time-interval` parameter (default 250 ms) tunes the queue batch timeout period. This timeout is applicable during periods of slower traffic and prevents events from being in a batch that is not filling quickly.

To set a higher batch size, ensure that you calculate the expected time it takes for a batch to fill during normal busy periods otherwise, the batch times out before filling. Estimate this value by dividing the total aggregate throughput rate (per second) by the number of deployed ingest grid nodes, and then dividing that number by the number of configured dispatchers (and, thus, queues). This provides the per-second throughput for each individual queue from which you can derive the appropriate batch timeout for any given batch size. In general, the less time events spend in the single hop router queues, the less the Java GC overhead there is for cleaning up events after transmission to the distribution grid.

Parallel queues and associated dispatchers

The `rti.ingestgrid.single-hop-gateway-sender.dispatcher` parameter (default threads(8)) configures the number of queues and associated dispatchers and socket connections for each route (that is, targeting each available distribution grid node). This tuning parameter is important to achieve sufficient grid-to-grid throughput, and to drive parallel processing on the remote distribution grid nodes. Each dispatcher thread directly drives a dedicated worker thread on a distribution node. If you increase the number of dispatcher threads, this increases the distribution grid's parallel processing. If you are experiencing queue backups in the single-hop router (evidenced by overflow-to-disk, warnings in the ingest grid GemFire log, and eventually backups into the ingester layer), then increase this value to attempt to alleviate the problem.

Each ingest grid node creates dispatcher threads based on the number of connections to each distribution grid node and the same number of worker threads on each distribution/service grid node. Do not increase this value too much or it will lead to excessive context switching on the distribution grid.

The total number of distribution grid worker threads is calculated by multiplying the number of ingest grid nodes times the configured number of dispatcher threads. For example, if the dispatcher threads value is 8, and the number of ingest grid nodes is 4, then a total of 32 worker threads will handle inbound events on each distribution grid node. This value must not exceed 2 times the number of available logical processing cores, as the threads mainly perform non-blocking operations and are busy as long as more events are arriving.

Maximum queue memory

The `rti.ingestgrid.single-hop-gateway-sender.maximum-queue-memory` parameter (default 40 MB) defines the maximum amount of memory each queue can consume before events start to overflow to disk. To maximize the use of memory to buffer events when the distribution grid is unable to keep up with traffic, this value must be configured as high as possible. When disk-overflow starts up, the throughput potential of the queues is dramatically reduced as each event must be written to file and potentially read back in before transmission.
If the backup occurs because of throughput overload, then this additional overhead makes the overload worse, causing more events to overflow to disk. Because there is still a small in-memory overhead for each queued event that overflows to disk, the system's memory consumption continues to increase, though at a slower rate. A sustained throughput overload might fill the memory up to the critical threshold, blocking all subsequent inbound traffic until the load in the queues can be lessened. This sequence of events usually occurs when the ingest grid experiences GemFire Resource Manager LowMemoryExceptions.

**Note:** Breaching maximum queue memory causes an increase at a fast rate. Set the value as high as possible because there is no better use for available memory under such conditions.

Because the ingest grid typically uses memory efficiently during normal processing, it is recommended that most of the heap be made available for queue memory to protect memory from filling up. A busy ingest grid process can survive on 5 GB of memory for all of its reference data and headroom for processing in-flight events. More than 5 GB is better. If the system runs low on memory, it will slow the system down due to more frequent Garbage Collection (GC); which will in turn cause more queueing; which will use more memory.

To calculate the optimal value for this parameter, divide the available extra memory (in MB) by the total number of queues to determine the maximum queue memory value. To calculate the total number of queues, multiply the dispatcher-threads value by the total number of distribution grid nodes.

For example, if you have 32 GB of memory allocated to the heap (or 32,000 MB), your dispatcher-threads value is set to 8, and 8 distribution grid nodes are deployed. That makes 8x8=64 total queues. Subtract 5 GB for working headroom from the total heap size and you get 27,000 MB remaining memory. Divide 27,000 by 64, and the maximum queue memory value is 400 MB (rounded down to the nearest 100).

**Tuning the number of buckets**

In RTI's GemFire-based partitioning, buckets are collections of application-level partitions, and are the unit of granularity used for balancing or re-balancing partitions across the available grid nodes due to cluster growth, data partition hot-spots, or HA failover events.

**rti.ingestgrid.pr.buckets (default 20)**

Configure enough buckets so that they can be evenly spread across the number of ingest grid instances in the cluster. If you plan to increase your cluster size without first redeploying the cluster from scratch, ensure that this value is high enough so that an approximately equal number of buckets can be taken from each of the existing instances to initialize the new instances with about the same number of buckets.

Set this value to at least 10 times the number of planned ingest grid nodes, leaving plenty of headroom for the cluster to grow and the buckets to be fairly evenly spread across all available nodes.

The default for "rti.ingestgrid.pr.buckets" has been set to 20 to reduce the memory overhead of processing in-flight events received from ingester client instances. Increase this value only for ingest grid cluster sizes larger than two. For larger clusters, configure a value that equates to ten buckets per ingest grid instance.
Tuning the distribution grid

This section describes how to tune the distribution grid’s flow control and parallelization. Figure 20 shows a sample distribution grid.

![Sample distribution grid diagram](image)

**Figure 20. Sample distribution grid**

The RTI distribution grid hosts the dual functions of distributing RTI output to downstream systems and performing several business functions.

**The role of the single-hop routing gateway hub**

The single-hop routing gateway hub is the component that receives all events from the ingest grid WAN single-hop router. It is a hub, because it receives batches from all of the upstream ingest grid instances, and because the implementation uses a GemFire component called a gateway hub to listen for ingest grid connections and to receive micro-batches of events.

*Note:* The bucket partitions on both grids are dynamically allocated the first time that grid starts up, and is automatically rebalanced in the background any time a new instance joins a running system. The bucket allocations happen independently on each of the grids, so each instance in the ingest grid typically processes events that map to at least one or two buckets hosted on each distribution grid node. This means, optimal single-hop routing of events is a many-to-many situation, and each distribution grid node receives connections and traffic from each ingest grid node.
Single-hop routing between the two grids assures that when an event arrives at a distribution grid node, no network I/O is necessary to fully complete the processing of the event. When you compare the processing and latency overhead of additional network I/O to the in-process or in-memory only logic execution, the difference is huge. Without the localized data benefits provide by this optimized routing, it would take significantly more hardware to process event flow.

The single-hop routing gateway hub encapsulates a GemFire CacheServer to listen for connections and to process inbound requests, making the starting point for processing basically the same as for the ingest grid. Each inbound connection from the upstream ingest grid instances to the local hub has its own dedicated processing thread, assuring that inbound batches never have to wait to be processed. As with the ingest grid, the processing model is to fully complete all work for every event in a batch before proceeding to process the next event, which is the most advantageous process for reducing Java GC overhead. After all batch events are processed, the gateway hub processing thread acknowledges the batch completion and sends an acknowledgement back to the sender before reading and processing the next batch.

As with the upstream ingester and ingest grid, the total throughput that a grid node can handle is limited by the time it takes to complete processing of a typical inbound batch and the number of parallel connections. If it takes half a second to complete processing a batch of 1,000 events, then each single-hop gateway hub thread can handle 2,000 events per second. To ensure that the distribution grid is fully utilized, ensure that there are enough parallel connections from the ingest grid feeding event micro-batches to the distribution grid. Tuning the number of parallel connections per server is done on the ingest grid using the rti.ingestgrid.single-hop-gateway-sender.dispatcher-threads property. For more information see Concurrent queues and handlers.

To calculate the total number of inbound connections for each distribution grid node, multiply the ingest grid rti.ingestgrid.single-hop-gateway-sender.dispatcher-threads property value times the total number of ingest grid instances. If the value is set to 8, and there are 4 ingest grid instances, then each distribution grid instance has $8 \times 4 = 32$ inbound connections and associated processing threads to perform parallel processing.

If you combine this information with the previous assumption that each thread can handle 2,000 events/second (and assuming there are sufficient hardware resources to scale from 1 to 32 threads), then the expected per-node throughput would be $32 \times 2,000 = 64,000$ events/second.

Thread context switching and other low-level sources of contention slightly reduce the per-thread throughput as more parallel processing threads are added, up to a point after which adding more reduces overall throughput (due to underlying hardware resource contention/exhaustion).

The highest possible throughputs typically achieved between one and two processing threads per available CPU core. Because there are other worker threads handing the output side of the distribution grid (as well as operating system threads we do not control), we recommend starting with a configuration that results in one processing thread per CPU core, and increasing this only if testing shows that CPU utilization has available headroom.
Chapter 13: Monitoring and Tuning Event Throughput

The distribution grid can scale both horizontally and vertically to handle large aggregate event throughput, so the focus is proper hardware sizing based on expected event throughput and optimal performance tuning.

Handling output files

When batched file output is enabled, write your own custom procedures for handling the contents of the files and then deleting them from the file system in a timely manner. The output rate can potentially be high, so even a generous amount of disk space can fill up quickly if external file processing stops for any length of time. Also keep in mind that very high rates of writing and deleting files may shorten the life of your disk hardware, particularly for SSD drive technology.

Ideally, build your file-handling processes to read and delete the RTI output files quickly before the operating system writes them to disk. This ensures that all of the I/O happens in memory, which is faster and more efficient than using physical disk. Work with your Linux operating system administrator to optimize the available page cache and related operating system tuning parameters to achieve this goal.

The role of LORS and LANDS filtering

All events are first fed into the LORS/LANDS opt-in manager to determine whether they are included based on their location, subscriber, or both. When doing performance tuning or planning on the distribution grid, it is important to first have an understanding of the percentage of events that will be go through LORS/LANDS, and the percentage that will be completely filtered out. For maximum efficiency, the goal is to filter out all traffic that is not of interest for the specific RTI uses and features you are using.

Note: Many customers configure RTI as a relay point for all events towards a downstream MPP database such as Greenplum, Hadoop, and so on. The opt-in manager can be configured to allow all traffic through by placing it in blacklist mode with no opt-outs loaded. (In blacklist mode, the meaning of opt-in is reversed to opt-out, and having no opt-outs means all traffic is allowed through).

Event processing

As each inbound event is handled, some local cache lookups take place to check location and subscriber opt-ins, and then the event is put through each relevant and enabled process flow. This can be a LORS, LANDS, LANDS-ESS-GeoLocation, and LORS-Metrics/Analytics process flow.

Possible outputs

The possible outputs of event processing are as follows:

- LORS/LANDS — By themselves, LORS/LANDS flows result in batched output to AMQP or file.
- LANDS-ESS — OUs provision ESS, and then external opt-ins, which are always linked to exactly one ESS. Events with subscribers externally opted-in for specific ESS are output to AMQP, once for each external opt-in. This behavior creates a multiplier effect where a single event might be routed to several output streams.
- LANDS-ESS-GeoLocation — If an event is opted-in for a specific ESS, its location is checked against any ESS, and linked to GeoFences by OUs. On a successful link, a notification is sent for each GeoLocation that matches an event’s cell tower or LAC location. Because GeoFences may overlap (actually overlap, or overlap within the uncertainly of an event’s location), this behavior can also create a
multiplier effect, where multiple output events are generated from a single input event.

- LORS-Metrics and Analytics — Depending on the configuration of the enabled metrics and analytics, an event can trigger various updates to cached metrics values and analytics calculation windows.

### Tuning the distribution output

As shown in Figure 20, all of the output from the distribution grid (to AMQP and to files) is micro-batched through reactor queues. Separate worker threads read and process the micro-batches from the reactor queues. The sections that follow describe the tuning property names for each specific outbound stream:

- Batch size and batch timeout
- File output compression

#### Batch size and batch timeout

Batch size and batch timeout properties work the same way as they do in other parts of the RTI system. Increasing the batch size and timeout can improve the overall throughput by increasing the payload size attached to each I/O operation. Batch timeout puts an upper limit on the average per-event processing latency by triggering any batches that take longer than the timeout value to reach the batch size.

For more information about the options available to tune batch size and timeout for the various output flows, refer to Tuning output flows.

#### File output compression

File output compression is available for all file output streams, and reduces the disk space used as a staging area. This reduction in the space used comes at the cost of the amount of CPU processing needed to perform the compression, as well as additional file I/O operations as the compressed version of the output events is written to disk.

To mitigate the overhead required for file compression, an option is also provided to configure compressionDelay. A longer delay is appropriate if you know that your procedures for handling the output files (again, externally provided) keeps up with the RTI output. This is the most efficient configuration to assign a value to compressionDelay to support fast output file processing. It ensures that these files are not compressed unless file handling procedures stop or slow down for some period of time and threaten to fill up file storage.

When reliably fast output file processing is not possible, a smaller compressionDelay value can improve overall efficiency by applying the file compression while the file contents are still in the Operating System Page Cache. This avoids physical disk I/O, which is far more expensive from a hardware resource utilization and latency perspective.

### Tuning output flows

The section describes the various property names for the specified flows. For more information about how batch size and timeout and file compression affect performance, refer to Batch size and batch timeout and File output compression.

- LORS File Out
Chapter 13: Monitoring and Tuning Event Throughput

- rti.distgrid.lors.file.batchSize
- rti.distgrid.lors.file.timeout
- rti.distgrid.lors.file.compress — Whether or not files are compressed
- rti.distgrid.lors.file.compressionDelay — Delay between file write and file compression in milliseconds

- LORS AMQP Out
  - rti.distgrid.lors.amqp.batchSize
  - rti.distgrid.lors.amqp.timeout

- LANDS File Out
  - rti.distgrid.lands.file.batchSize
  - rti.distgrid.lands.file.timeout
  - rti.distgrid.lands.file.compress — Whether or not files are compressed
  - rti.distgrid.lands.file.compressionDelay — Delay between file write and file compression in milliseconds

- LANDS AMQP Out
  - rti.distgrid.lands.amqp.batchSize
  - rti.distgrid.lands.amqp.timeout

- LANDS-ESS AMQP: The following also apply to ESS-GeoLocation AMQP output.
  - rti.distgrid.ess.batch.size
  - rti.distgrid.ess.batch.timeout
  - rti.distgrid.ess.compressPayload — Whether or not to apply compression to the AMQP payload. Compression reduces network I/O at the expense of increased processing overhead. This is useful when network bandwidth is near or at capacity, but processing load is not.

**Tuning the number of buckets to rebalance partitions**

The `rti.distgrid.pr.buckets` option used to tune the number of buckets applies the same way to both the ingest grid and the distribution grid. In GemFire partitioning, buckets are collections of application-level partitions, and are the unit-of-granularity used for balancing and re-balancing partitions across the available grid nodes due to cluster growth, data partition hot spots, or high availability failover events.

The `rti.distgrid.pr.buckets` parameter (default value 37) must be configured for enough buckets to be evenly spread across the number of ingest grid instances in the cluster. If you plan to increase your cluster size without having to first re-deploy the cluster from scratch, this value must be high enough so that an approximately equal number of buckets can be taken from each of the existing instances to initialize the new instances with about the same number of buckets.
Set this value to at least 10 times the number of planned distribution nodes, which leaves plenty of headroom for the cluster to grow and for the buckets to still be fairly evenly spread across all available nodes.

**Stateful analytics and metrics**

If you are using the stateful analytics or metrics features, more memory is needed to cache and handle memory turnover for the various event data and mathematical aggregates.

The additional overhead required varies depending on:
- The number of enabled categories
- The configured frequency of distribute, save/, and export operations
- The configured number of leading and lagging windows (to handle out of order events)
- Window size

If you are enabling metrics or stateful analytics for the first time in an existing RTI deployment, it is important to measure the impact of these features carefully before deploying them into production, because they will likely add a significant amount of processing overhead. We recommend that you consult with RTI Support or your RTI Field Engineering representative to help understand the detailed performance impacts and to plan the roll-outs of these features.

**Output stream filtering**

When configuring JSON-based output stream filters, the "where" filtering expression is specified in SpEL. Each additional expression within the SpEL filter adds processing overhead, so keep “output stream filter where” expressions as simple as possible.

When enabling output stream filtering in RTI, measure the impact of new and changed SpEL filter expressions, and adjust the system’s sizing if necessary.
# Setting event threshold

When the buffer size reaches a specified threshold, you can drop incoming events.

## Event threshold

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Editable?</th>
</tr>
</thead>
<tbody>
<tr>
<td>BatchCount</td>
<td>The number of batches received by this batcher.</td>
<td>No</td>
</tr>
<tr>
<td>BatchSize</td>
<td>The maximum size of a batch before the ReactorStreamBatchingDelegatingHandler Mbean pushes it.</td>
<td>No</td>
</tr>
<tr>
<td>BufferOverrun</td>
<td>Boolean value that informs the system to generate an alert when ProbableBufferSize exceeds MaxBufferSize.</td>
<td>No</td>
</tr>
<tr>
<td>DroppedCount</td>
<td>The current number of dropped messages.</td>
<td>No</td>
</tr>
<tr>
<td>MaxBatchSize</td>
<td>The size of the maximum batch received.</td>
<td>No</td>
</tr>
<tr>
<td>MaxBufferSize</td>
<td>The maximum buffer size before the system considers the buffer to be in an overrun state.</td>
<td>Yes</td>
</tr>
<tr>
<td>MeanBatchSize</td>
<td>The calculated mean size of received batches.</td>
<td>No</td>
</tr>
<tr>
<td>MinBatchSize</td>
<td>The size of the minimum batch received.</td>
<td>No</td>
</tr>
<tr>
<td>ProbableBufferSize</td>
<td>The estimated current size of the buffer.</td>
<td>No</td>
</tr>
<tr>
<td>Timeout</td>
<td>The amount of time remaining before a batch will be sent.</td>
<td>No</td>
</tr>
</tbody>
</table>

Table 35. Event threshold MBean

<table>
<thead>
<tr>
<th>Domain</th>
<th>io.pivotal.rti.ingester.adr.processor</th>
</tr>
</thead>
<tbody>
<tr>
<td>MBean type</td>
<td>ReactorStreamBatchingDelegatingHandler</td>
</tr>
<tr>
<td>MBean name</td>
<td>reactorGemfireBatcher</td>
</tr>
<tr>
<td>Java class</td>
<td>com.sun.proxy.$Proxy29</td>
</tr>
</tbody>
</table>
Chapter 14 Configuration Properties

This chapter presents the following topics:

**Referencing configuration properties** ................................................................. 237
**Common properties** ............................................................................................ 237
**Protocol adapter properties** ................................................................................ 240
**GemFire cache server properties** .......................................................................... 242
**GemFire pool properties** ..................................................................................... 243
**GemFire disk store properties** ............................................................................... 244
**Ingester properties** ............................................................................................ 245
**Distribution grid properties** ................................................................................ 250
**Provisioning Manager and RabbitMQ properties** ............................................. 265
**Event format properties** ..................................................................................... 267
**RDL properties** .................................................................................................. 270
**REST API client properties** ................................................................................ 271
**HDFS properties** ............................................................................................... 271
Configuration properties

This section names and describes the properties that you can configure for different RTI components, including environment (env) properties. Some properties are common to all components, some are used by multiple components, and some are specific to one component.

To set properties, use the following commands:

```
rtish> rti set property --instances instance --name <name> --value <value>
rtish> rti set env --instances instance --name <name> --value <value>
```

For example:

```
rtish> rti set property --instances ingest-grid-locator@* --name rti.locator.port --value 10334
```

Common properties

The following properties are used by some or all of the RTI components.

<table>
<thead>
<tr>
<th>Property name</th>
<th>Default value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rti.anonymizer.encodeExtraInIMEI</td>
<td>true</td>
<td></td>
</tr>
<tr>
<td>rti.anonymizer.formatterForIMEIMask</td>
<td>false</td>
<td></td>
</tr>
<tr>
<td>rti.anonymizer.includeLength</td>
<td>false</td>
<td>Include the length by appending it to the hashed string.</td>
</tr>
<tr>
<td>rti.anonymizer.keepMsisdnCountryCode</td>
<td>false</td>
<td>Keep the country code in the anonymized MSISDN.</td>
</tr>
<tr>
<td>rti.anonymizer.lengthDelimiter</td>
<td></td>
<td>Delimiter character used in anonymized strings, when the length is appended to the end.</td>
</tr>
<tr>
<td>rti.anonymizer.noHashCalledDigits</td>
<td>false</td>
<td>Whether to hash the called digits.</td>
</tr>
<tr>
<td>rti.anonymizer.noHashCallingDigits</td>
<td>false</td>
<td>Whether to hash the calling digits.</td>
</tr>
<tr>
<td>rti.anonymizer.noHashIMSI</td>
<td>false</td>
<td>Whether to hash the IMSI.</td>
</tr>
<tr>
<td>rti.anonymizer.noHashMSISDN</td>
<td>false</td>
<td>Whether to hash the MSISDN.</td>
</tr>
<tr>
<td>rti.anonymizer.noMaskIMEI</td>
<td>false</td>
<td>Whether to hash the IMEI.</td>
</tr>
<tr>
<td>rti.anonymizer.preserveLength</td>
<td>false</td>
<td>Preserve the length by appending it to the hashed string.</td>
</tr>
<tr>
<td>rti.gemfire.bind-address</td>
<td>N/A</td>
<td>The IP address that GemFire will use to communicate in the grid.</td>
</tr>
</tbody>
</table>
### Chapter 14: Configuration Properties

<table>
<thead>
<tr>
<th>Property name</th>
<th>Default value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rti.gemfire.distributedsystemid</td>
<td>N/A</td>
<td>For all components connected to the ingest grid: 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>For all components connected to the distribution grid: 2</td>
</tr>
<tr>
<td>rti.gemfire.locators</td>
<td>localhost[10334]</td>
<td>Locators available for connections, defined as host[port]. For example: 127.0.0.1[10335], 10.0.0.101[10334], 10.0.0.102[10334]</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Note:</strong> It is critically important that all application instances belonging to a grid family share the same value for this property. This means that all ingester, ingest grid, and locators servicing them share the same identical value, and that all Provisioning Manager, distribution grid, and locators servicing them share the same identical value. Additionally, the locators for distribution grid and ingest grid deployments can never be shared.</td>
</tr>
<tr>
<td>rti.gfe.critical.heap</td>
<td>80</td>
<td>A JVM memory management property: threshold at which RTI servers stop accepting data updates and revert to read-only mode (until available heap falls below the threshold due to GC activity). Recommended value is 90.</td>
</tr>
<tr>
<td>rti.gfe.eviction.heap</td>
<td>70</td>
<td>A JVM memory management property: Java heap utilization threshold at which overflow-to-disk eviction is initiated for subscriber events and reference data. Set this value the same as, or slightly higher than, the following Java CMS value: XX:CMSInitiatingOccupancyFraction</td>
</tr>
<tr>
<td>rti.hasher.algorithm</td>
<td>sha256</td>
<td>The hashing algorithm used by anonymizer processes.</td>
</tr>
<tr>
<td>rti.optin.blacklist.location.disabled</td>
<td>false</td>
<td>Disable checking for blacklists-based location.</td>
</tr>
<tr>
<td>rti.optin.blacklist.location.keyoptimization</td>
<td>false</td>
<td>Use an optimization for location key lookups when checking location blacklists.</td>
</tr>
<tr>
<td>rti.optin.blacklist.subscriber.disabled</td>
<td>false</td>
<td>Disable checking for opt- ins-based subscriber.</td>
</tr>
</tbody>
</table>
### Chapter 14: Configuration Properties

<table>
<thead>
<tr>
<th>Property name</th>
<th>Default value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rti.optin.blacklist.subscriber.keyoptimization</td>
<td>false</td>
<td>Use an optimization for subscriber key lookups when checking subscriber blacklists.</td>
</tr>
<tr>
<td>rti.optin.global.location.disabled</td>
<td>false</td>
<td>Disable checking for global opt-ins-based location.</td>
</tr>
<tr>
<td>rti.optin.global.location.keyoptimization</td>
<td>false</td>
<td>Use an optimization for location key lookups when checking global opt-in locations.</td>
</tr>
<tr>
<td>rti.optin.global.subscriber.disabled</td>
<td>false</td>
<td>Disable checking for opt-ins-based subscriber.</td>
</tr>
<tr>
<td>rti.optin.global.subscriber.keyoptimization</td>
<td>false</td>
<td>Use an optimization for subscriber key lookups when checking global opt-in subscribers.</td>
</tr>
<tr>
<td>rti.salt</td>
<td>N/A</td>
<td>A 'seed' that is mixed into the data as the first step of the anonymization process.</td>
</tr>
<tr>
<td>rti.startup.retry.interval</td>
<td>0</td>
<td>Time (milliseconds) that an application, after failing to start, will wait before trying to start again. If set to 0, the application will not try to restart. Default value is 0. Only applicable to the locator, ingestgrid, and distgrid applications.</td>
</tr>
<tr>
<td>rti.trace.device.disable</td>
<td>false</td>
<td>Disable trace by device.</td>
</tr>
<tr>
<td>rti.trace.disable</td>
<td>false</td>
<td>Disable trace (master switch).</td>
</tr>
<tr>
<td>rti.trace.location.disable</td>
<td>false</td>
<td>Disable trace by location.</td>
</tr>
<tr>
<td>rti.trace.subscriber.disable</td>
<td>false</td>
<td>Disable trace by subscriber.</td>
</tr>
</tbody>
</table>

See the GemFire documentation for information about the following properties, which are set in the GemFire distributed system on startup: [http://gemfire.docs.pivotal.io/index.html](http://gemfire.docs.pivotal.io/index.html)

- rti.gemfire.archive-disk-space-limit=10000
- rti.gemfire.archive-file-size-limit=100
- rti.gemfire.conserve-sockets=false
- rti.gemfire.enable-network-partition-detection=false
- rti.gemfire.enable-time-statistics=false
- rti.gemfire.license-application-cache=
- rti.gemfire.license-server-timeout=2000
- rti.gemfire.license-working-dir=${rti.conf}
- rti.gemfire.log-disk-space-limit=10000
- rti.gemfire.log-file-size-limit=100
- rti.gemfire.log-file=${rti.logs}/rti-ingestgrid-gemfire.log
Protocol adapter properties

The properties in this section apply to the RTI application types that parse raw input event payloads. These are:

- **Ingestor** -- At least, parses the header fields of each event to drive downstream event routing (typically using IMSI), as well as to obtain fields needed to drive any ingester-enabled filters (Location, IMSI).
- **Ingest grid** -- Parses all events and transforms them into RTI's normalized event model upon receipt by the ingest grid. Thereafter, RTI logic only interacts with the normalized events.
- **Syphon** -- Taps event streams as they arrive in the ingest grid (and before normalization), and parses raw events before outputting them in JSON format.

Only properties related to the parsing of raw protocol events are described in this section. Other protocol-specific events exist for non-parsing purposes (for example, connectivity or store-and-forward queue tuning), and are not applicable to more than one application type. Properties not specific to parsing are discussed in their respective application type's dedicated Properties reference.

Additionally, there are no specific references in this section to proprietary protocol adapter implementation. Details for these are separately provided to licensed customers only.

**IMPORTANT:** The values of all properties in this section MUST BE configured the same way for all of the application types. Make the property assignment once for all affected instances. For example, if you are configuring the version of adr, the RTISH command might look like this:

```
rti set property --instances ingest1@host1,ingest-grid1@host2,syphon-app@host2
    --name <protocol>.maxFrameLength --value 65536
```

This approach is highly recommended as it is less error prone than using separate assignment statements for each application instance.

In Table 37, `<protocol>` may be one of the following values, or the protocol name of a custom or proprietary adapter:

- adr
- gb
iucs
iups
s1mme
gngi

Additionally, proprietary adapter implementations are labeled with only "p-adapter". The actual property name depends on the specific proprietary adapter type, as separately documented and provided as-needed to licensed customers.

Most of the common protocol parsing properties do not apply to radius.

Table 37. Protocol adapter properties

<table>
<thead>
<tr>
<th>Property name</th>
<th>Default value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>adr.&lt;p-adapter&gt;.version</td>
<td>NA</td>
<td>Refer to separately provided documents for the list of supported values. (The radius protocol supports a single version, which is rfc2865, rfc2866-based.)</td>
</tr>
<tr>
<td>gb.&lt;p-adapter&gt;.version</td>
<td>NA</td>
<td>Refer to separately provided documents for the list of supported values.</td>
</tr>
<tr>
<td>iucs.&lt;p-adapter&gt;.version</td>
<td>NA</td>
<td>Refer to separately provided documents for the list of supported values.</td>
</tr>
<tr>
<td>iups.&lt;p-adapter&gt;.version</td>
<td>NA</td>
<td>Refer to separately provided documents for the list of supported values.</td>
</tr>
<tr>
<td>gngi.&lt;p-adapter&gt;.version</td>
<td>NA</td>
<td>Refer to separately provided documents for the list of supported values.</td>
</tr>
<tr>
<td>s1mme.&lt;p-adapter&gt;.version</td>
<td>NA</td>
<td>Refer to separately provided documents for the list of supported values.</td>
</tr>
<tr>
<td>gngi.nativeCodec</td>
<td>true</td>
<td>Controls whether the native codec is used for gngi.</td>
</tr>
<tr>
<td>s1mme.nativeCodec</td>
<td>true</td>
<td>Controls whether the native codec is used for s1mme.</td>
</tr>
<tr>
<td>&lt;protocol&gt;.enabled</td>
<td>true</td>
<td>Controls whether the individual protocol is enabled. The s1mme and gngi protocols default to false. This property also applies to the radius protocol, which defaults to false. The other common properties do not apply to radius.</td>
</tr>
<tr>
<td>&lt;protocol&gt;.maxFrameLength</td>
<td>65536</td>
<td>Maximum frame length for the protocol adapter's TCP server listener.</td>
</tr>
</tbody>
</table>
### Configuration Properties

<table>
<thead>
<tr>
<th>Property name</th>
<th>Default value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;protocol name&gt;.dropInitialImage</code></td>
<td>false</td>
<td>Some source systems send a large batch of data in the initial connection that can cause some backpressure in RTI. To reduce the effect of this backpressure on the source system, instruct RTI to drop the initial image received from the source system. A boolean that tells RTI to drop the initial image. For example: <code>rti set property --instances ingester-one@* --name adr.dropInitialImage --value true</code></td>
</tr>
<tr>
<td><code>&lt;protocol name&gt;.dropInitialImageSize</code></td>
<td>200000</td>
<td>Some source systems send a large batch of data in the initial connection, which can cause some backpressure in RTI. To reduce the effect of this backpressure on the source system, instruct RTI to drop a specific number of events in the initial image received from the source system. A long value specifying the number of events to drop. For example: <code>rti set property --instances ingester-one@* --name adr.dropInitialImageSize --value 100000</code></td>
</tr>
</tbody>
</table>

### GemFire cache server properties

These cache server properties apply to both the ingest grid and the distribution grid.

**Table 38. GemFire server properties**

<table>
<thead>
<tr>
<th>Property name</th>
<th>Default value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>rti.cache-server.max-threads</code></td>
<td>20</td>
<td>Java native I/O thread pool size on a server that services inbound client requests (ingester, RDL, or Provisioning Manager).</td>
</tr>
<tr>
<td><code>rti.cache-server.max-connections</code></td>
<td>800</td>
<td>Maximum number of allowed concurrent client connections across all connected clients (each client's pool typically has one connection per active thread).</td>
</tr>
<tr>
<td><code>rti.cache-server.bind-address</code></td>
<td></td>
<td>Network interface to use for client/server communication: incoming connections from RTISH, the RDL, and the Provisioning Manager. An empty value means bind to all network interfaces. Segregating client/server, peer-to-peer, and WAN gateway traffic onto different NICs is recommended.</td>
</tr>
</tbody>
</table>
### Property name

<table>
<thead>
<tr>
<th>Property name</th>
<th>Default value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>rti.cache-server.port</code></td>
<td>0</td>
<td>Specific TCP port number, set only if needed for client/server communication (due to a firewall). Used for incoming connections from RTISH, the Reference Data Loader, and the Provisioning Manager.</td>
</tr>
<tr>
<td><code>rti.cache-server.hostname-for-clients</code></td>
<td></td>
<td>Specific host name for clients, set only if the bind-address used by the cache server is not visible to the client. Any value set here is used by the client as the destination address for server connectivity. The value is provided to clients via the locators when the client initiates a connection request. The value of this property is an IPv4 address (dotted quad) or hostname (not recommended) that the clients can resolve and connect to from RTISH, the RDL, and the Provisioning Manager.</td>
</tr>
</tbody>
</table>

### GemFire pool properties

Recommended values for these properties may vary by RTI component. These properties are used for GemFire client tuning by the ingester, Reference Data Loader, and Provisioning Manager.

#### Table 39. GemFire pool properties

<table>
<thead>
<tr>
<th>Property name</th>
<th>Ingestor value</th>
<th>PM value</th>
<th>RDL value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>rti.&lt;component&gt;.gemfire.pool.freeConnectionTimeout</code></td>
<td>10000</td>
<td>10000</td>
<td>5000</td>
</tr>
<tr>
<td><code>rti.&lt;component&gt;.gemfire.pool.idleTimeout</code></td>
<td>5000</td>
<td>25000</td>
<td>25000</td>
</tr>
<tr>
<td><code>rti.&lt;component&gt;.gemfire.pool.keepAlive</code></td>
<td>false</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td><code>rti.&lt;component&gt;.gemfire.pool.loadConditioningInterval</code></td>
<td>300000</td>
<td>300000</td>
<td>600000</td>
</tr>
<tr>
<td><code>rti.&lt;component&gt;.gemfire.pool.maxConnections</code></td>
<td>64</td>
<td>10</td>
<td>-1</td>
</tr>
<tr>
<td><code>rti.&lt;component&gt;.gemfire.pool.minConnections</code></td>
<td>64</td>
<td>1</td>
<td>8</td>
</tr>
<tr>
<td><code>rti.&lt;component&gt;.gemfire.pool.multiUserAuthentication</code></td>
<td>false</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td><code>rti.&lt;component&gt;.gemfire.pool.pingInterval</code></td>
<td>10000</td>
<td>10000</td>
<td>30000</td>
</tr>
<tr>
<td><code>rti.&lt;component&gt;.gemfire.pool.prSingleHopEnabled</code></td>
<td>true</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td><code>rti.&lt;component&gt;.gemfire.pool.readTimeout</code></td>
<td>10000</td>
<td>10000</td>
<td>20000</td>
</tr>
<tr>
<td><code>rti.&lt;component&gt;.gemfire.pool.retryAttempts</code></td>
<td>-1</td>
<td>-1</td>
<td>2</td>
</tr>
<tr>
<td><code>rti.&lt;component&gt;.gemfire.pool.socketBufferSize</code></td>
<td>1572864</td>
<td>1572864</td>
<td>524288</td>
</tr>
<tr>
<td><code>rti.&lt;component&gt;.gemfire.pool.statisticInterval</code></td>
<td>-1</td>
<td>-1</td>
<td>5000</td>
</tr>
</tbody>
</table>
**GemFire disk store properties**

These properties apply to both the ingest grid and the distribution grid. GemFire disk stores provide a means of persisting data to disk as a backup to the in-memory cache. When the system restarts, the data can be recovered from the disk stores. Data may also overflow to disk stores when the system starts to run low on memory. For more details about disk store configuration, see *Designing and Configuring Disk Stores* in the GemFire documentation.

| Property name                                      | Default value | Description                                                                 |
|---------------------------------------------------|---------------|                                                                            |
| rti.gfe.disk-store.auto-compact.enabled           | true          | Enables or disables automatic compaction of disk stores.                   |
| rti.gfe.disk-store.compaction-threshold           | 50            | Sets the threshold at which an operation log file (oplog) will become compactable. Until it reaches this threshold, an oplog is not compacted. The threshold value is a percentage in the range 0 to 100. |
| rti.gfe.disk-store.max-size                       | 655360        | Sets the maximum size of the disk stores. Adjust this value if the expected reference data sets (per RTI installation) exceed 655GB. For example, a large volume of external subscriber opt-ins could cause the reference data disk store to run out of space. |
These properties are specific to locator components.

### Table 41. Locator properties

<table>
<thead>
<tr>
<th>Property name</th>
<th>Default value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rti.locator.host</td>
<td></td>
<td>The IP address to bind the location to. Use an IP address accessible to both grid nodes and the clients.</td>
</tr>
<tr>
<td>rti.locator.port</td>
<td>10334</td>
<td>Port to run the locator on.</td>
</tr>
<tr>
<td>rti.gemfire.remote-locators</td>
<td>None</td>
<td>For ingest grid locators, set this property to point to the locators in the distribution grid. For example: 10.0.0.101[10334], 10.0.0.102[10334]</td>
</tr>
</tbody>
</table>

**Note:** If you need to configure undocumented GemFire properties for locator applications, contact technical support.

---

### Ingestor properties

In this table, `<protocol>` is one of the following values:

- adr
- gb
- iucs
- iups
- s1mme
- gngi

Most of the common protocol properties do not apply to radius.

Many of the RTI properties listed under Common Properties are used by the ingester and other components.

**Note:** All properties specific to parsing are covered in the section Protocol adapter properties.

### Table 42. Ingestor configuration properties

<table>
<thead>
<tr>
<th>Property name</th>
<th>Default value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rti.gemfire.locators</td>
<td>localhost[10334]</td>
<td>The GemFire Locator to connect to.</td>
</tr>
<tr>
<td>rti.gemfire.log-file</td>
<td>${rti.logs}/rti-ingester-gemfire.log</td>
<td></td>
</tr>
<tr>
<td>Property name</td>
<td>Default value</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------------------------------</td>
<td>----------------------------------------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>rti.gemfire.statistic-archive-file</td>
<td>${rti.logs}/rti-ingester-gemfire-stats.gfs</td>
<td></td>
</tr>
<tr>
<td>rti.gemfire.archive-disk-space-limit</td>
<td>10000</td>
<td></td>
</tr>
<tr>
<td>rti.gemfire.archive-file-size-limit</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>rti.gemfire.bind-address</td>
<td></td>
<td></td>
</tr>
<tr>
<td>rti.gemfire.conserve-sockets</td>
<td>FALSE</td>
<td></td>
</tr>
<tr>
<td>rti.gemfire.distributed-system-id</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>rti.gemfire.enable-network-partition-detection</td>
<td>FALSE</td>
<td></td>
</tr>
<tr>
<td>rti.gemfire.enable-time-statistics</td>
<td>FALSE</td>
<td></td>
</tr>
<tr>
<td>rti.gemfire.log-disk-space-limit</td>
<td>10000</td>
<td></td>
</tr>
<tr>
<td>rti.gemfire.log-file-size-limit</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>rti.gemfire.log-level</td>
<td>config</td>
<td></td>
</tr>
<tr>
<td>rti.gemfire.security-log-level</td>
<td>config</td>
<td></td>
</tr>
<tr>
<td>rti.gemfire.statistic-sample-rate</td>
<td>5000</td>
<td></td>
</tr>
<tr>
<td>rti.gemfire.statistic-sampling-enabled</td>
<td>TRUE</td>
<td></td>
</tr>
<tr>
<td>rti.hasher.algorithm</td>
<td>sha256</td>
<td></td>
</tr>
<tr>
<td>rti.ingester.trace.spel</td>
<td></td>
<td></td>
</tr>
<tr>
<td>rti.ingester.gemfire.pool.freeConnection</td>
<td>10000</td>
<td></td>
</tr>
<tr>
<td>rti.ingester.gemfire.pool.idleTimeout</td>
<td>-1</td>
<td></td>
</tr>
<tr>
<td>rti.ingester.gemfire.pool.keepAlive</td>
<td>FALSE</td>
<td></td>
</tr>
<tr>
<td>rti.ingester.gemfire.pool.loadConditioningInterval</td>
<td>30000</td>
<td></td>
</tr>
<tr>
<td>rti.ingester.gemfire.pool.minConnections</td>
<td>32</td>
<td></td>
</tr>
<tr>
<td>rti.ingester.gemfire.pool.maxConnections</td>
<td>800</td>
<td></td>
</tr>
<tr>
<td>rti.ingester.gemfire.pool.multiUserAuthentication</td>
<td>FALSE</td>
<td></td>
</tr>
<tr>
<td>rti.ingester.gemfire.pool.name</td>
<td>DEFAULT</td>
<td></td>
</tr>
<tr>
<td>rti.ingester.gemfire.pool.pingInterval</td>
<td>10000</td>
<td></td>
</tr>
<tr>
<td>rti.ingester.gemfire.pool.prSingleHopEnabled</td>
<td>TRUE</td>
<td></td>
</tr>
<tr>
<td>rti.ingester.gemfire.pool.readTimeout</td>
<td>10000</td>
<td></td>
</tr>
<tr>
<td>rti.ingester.gemfire.pool.retryAttempts</td>
<td>-1</td>
<td></td>
</tr>
<tr>
<td>Property name</td>
<td>Default value</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------------------------------</td>
<td>---------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>rti.ingester.gemfire.pool.statistic Interval</td>
<td>-1</td>
<td></td>
</tr>
<tr>
<td>rti.ingester.gemfire.pool.subscription Enabled</td>
<td>TRUE</td>
<td></td>
</tr>
<tr>
<td>rti.ingester.gemfire.pool.socketBufferSize</td>
<td>1572864</td>
<td></td>
</tr>
<tr>
<td>rti.ingester.gemfire.pool.serverGroup</td>
<td></td>
<td></td>
</tr>
<tr>
<td>rti.ingester.gemfire.pool.threadLocal Connections</td>
<td>FALSE</td>
<td></td>
</tr>
<tr>
<td>rti.ingester.metrics.pollRate</td>
<td>60</td>
<td>Specifies how frequently JMX metrics from this node are published into the system</td>
</tr>
<tr>
<td>rti.ingester.enrichers.anonymizer</td>
<td>TRUE</td>
<td>Anonymizes user-identifiable fields in each ProtocolEvent</td>
</tr>
<tr>
<td>rti.ingester.filters.imsi</td>
<td>TRUE</td>
<td>Enables the IMSI filter: drops events with no IMSI.</td>
</tr>
<tr>
<td>rti.ingester.filters.eventTime</td>
<td>TRUE</td>
<td>Enables the event time filter: drops events with no event time or bad event time.</td>
</tr>
<tr>
<td>rti.ingester.filters.location</td>
<td>TRUE</td>
<td>Enables the location filter: drops events with no location.</td>
</tr>
<tr>
<td>rti.ingester.{protocol}.batch.size</td>
<td>8192</td>
<td>Specifies the batch size tuning for this protocol. Maximum size is 9999.</td>
</tr>
<tr>
<td>rti.ingester.{protocol}.batch.timeout</td>
<td>4000</td>
<td>Specifies the batch timeout tuning for this protocol.</td>
</tr>
<tr>
<td>rti.ingester.{protocol}.batch.backlog</td>
<td>65536</td>
<td>Specifies the batcher backlog tuning for this protocol.</td>
</tr>
<tr>
<td>rti.ingester.{protocol}.batch.threads</td>
<td>16</td>
<td>Specifies the batcher threadpool size tuning for this protocol.</td>
</tr>
<tr>
<td>rti.ingester.active</td>
<td>FALSE</td>
<td>When FALSE, the ingester accepts input but is not connected to the ingest grid. Useful for observing input data rates and doing system restarts.</td>
</tr>
<tr>
<td>{protocol}.tcp.port</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• adr -- 29001</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• gb -- 29002</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• iucs -- 29003</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• iups -- 29004</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• s1mme -- 29006</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• gngi -- 29007</td>
</tr>
<tr>
<td>{protocol}.tcp.host</td>
<td>0.0.0.0</td>
<td>Specifi specific address to bind the listener to (if any).</td>
</tr>
<tr>
<td></td>
<td>(All interfaces)</td>
<td></td>
</tr>
</tbody>
</table>
These properties are specific to the ingest grid.

### Table 43. Ingest grid properties

<table>
<thead>
<tr>
<th>Property name</th>
<th>Default value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rti.indexes.geoSpatial.startupRebuildDelayMillis</td>
<td>1000</td>
<td>The frequency with which in-memory geospatial indexes associated with the cell tower data are rebuilt (in milliseconds).</td>
</tr>
<tr>
<td>rti.ingestgrid.data.expiry.seconds</td>
<td>90</td>
<td>Time-to-live for ingest grid subscriber event data.</td>
</tr>
<tr>
<td>rti.ingestgrid.filters.imsi.enabled</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>rti.ingestgrid.filters.eventTime.enabled</td>
<td></td>
<td></td>
</tr>
<tr>
<td>rti.ingestgrid.filters.cellTower.enabled</td>
<td></td>
<td></td>
</tr>
<tr>
<td>rti.ingestgrid.filters.blacklist.enabled</td>
<td></td>
<td></td>
</tr>
<tr>
<td>rti.ingestgrid.filters.location.keyoptimization</td>
<td></td>
<td></td>
</tr>
<tr>
<td>rti.ingestgrid.filters.blacklist.subscriber.keyoptimization</td>
<td></td>
<td></td>
</tr>
<tr>
<td>rti.ingestgrid.filters.blacklist.location.disabled</td>
<td></td>
<td></td>
</tr>
<tr>
<td>rti.ingestgrid.filters.blacklist.subscriber.disabled</td>
<td></td>
<td></td>
</tr>
<tr>
<td>rti.ingestgrid.filters.blacklist.location.keyoptimization</td>
<td></td>
<td></td>
</tr>
<tr>
<td>rti.ingestgrid.filters.blacklist.subscriber.keyoptimization</td>
<td></td>
<td></td>
</tr>
<tr>
<td>rti.ingestgrid.filters.blacklist.location.disabled</td>
<td></td>
<td></td>
</tr>
<tr>
<td>rti.ingestgrid.filters.globalLocationOptin'useStartTimeForOptin</td>
<td></td>
<td></td>
</tr>
<tr>
<td>rti.ingestgrid.filters.globalLocationOptin.enabled</td>
<td></td>
<td></td>
</tr>
<tr>
<td>rti.ingestgrid.filters.globalSubscriberOptin'useStartTimeForOptin</td>
<td></td>
<td></td>
</tr>
<tr>
<td>rti.ingestgrid.filters.globalSubscriberOptin.enabled</td>
<td></td>
<td></td>
</tr>
<tr>
<td>rti.ingestgrid.filters.blacklist.anonymiseMode</td>
<td>true</td>
<td>Controls the behavior of the blacklist filter. If true, events are allowed through, but with all identifying fields replaced with random values. If false, then the events are discarded.</td>
</tr>
<tr>
<td>rti.ingestgrid.geocodedata.expiry.seconds</td>
<td>90</td>
<td>Time-to-live for ingest grid geocode data.</td>
</tr>
<tr>
<td>Property name</td>
<td>Default value</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------------------</td>
<td>---------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>rti.ingestgrid.enricher.imsiTmsi</td>
<td>false</td>
<td>The mobile network always registers a TMSI+IMSI pair of values when a device connects using a cell tower, but the IMSIs are not maintained in the event until the device connects again. RTI keeps a persistent store of both values and is capable of reverse mapping the TMSI to its IMSI. This property enables or disables the enricher that maps IMSIs to TMSIs. When this property is enabled and an event with a TMSI and IMSI pair of values is ingested, that mapping is persisted during event processing. When an event with a TMSI but no IMSI is ingested, the IMSI is retrieved from the persisted data store and added back to the event. (Events without IMSIs are dropped.) See also the expiry time controls for the mapping (ingester and ingester Grid).</td>
</tr>
<tr>
<td>rti.ingestgrid.imsiTmsiMapping.expiry.seconds</td>
<td>7200</td>
<td>Time-to-live for IMSI-to-TMSI mappings. For maximum effect, configure the expiry to be a little longer than the average time a user is permitted to use the same TMSI/IMSI pairing.</td>
</tr>
<tr>
<td>rti.ingestgrid.ingester.enabled</td>
<td>true</td>
<td>Configures the ingest grid process to instantiate a collocated ingester as part of the deployment. When a separate stand-alone ingester process is desired, the value of this property must be set to false and the ingester process must be defined/configured separately.</td>
</tr>
<tr>
<td>rti.ingestgrid.pr.buckets</td>
<td>20</td>
<td>The number of partitioned region buckets to use. There should be at least 10 times the number of buckets as server instances. The default has been set to 20 to reduce the memory overhead of processing in-flight events received from ingester Client instances. This value should only be increased for ingest grid cluster sizes larger than at least 2. For larger clusters, we recommend configuring a value that divides to 10 buckets per ingest grid instance.</td>
</tr>
</tbody>
</table>
### Chapter 14: Configuration Properties

#### Configuration Properties

<table>
<thead>
<tr>
<th>Property name</th>
<th>Default value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rti.ingestgrid.pr.redundancy</td>
<td>0</td>
<td>The number of redundant copies of data to keep in memory across different server instances.</td>
</tr>
<tr>
<td>rti.ingestgrid.protocol.data.eviction.ttl</td>
<td>5</td>
<td>Time-to-live for protocol event region data, in seconds (should be set low since these regions are only used as event conduits from the client ingester to ingester server processing logic).</td>
</tr>
<tr>
<td>rti.gateway.socket-buffer-size</td>
<td>4194304</td>
<td>Common to both grids, and should be set the same on both grids (not setting them the same could degrade performance).</td>
</tr>
<tr>
<td>rti.gateway.socket-read-timeout</td>
<td>10000</td>
<td>The time that the IG Gateway Sender will wait for an ack from the DG GW Receiver before timing out a connection. <strong>Set to 60000 if you notice connections bouncing periodically.</strong></td>
</tr>
<tr>
<td>rti.gemfire.remote-locators</td>
<td>None</td>
<td>For ingest grid locators, set this property to point to the locators in the distribution grid. For example: 10.0.0.101[10334], 10.0.0.102[10334].</td>
</tr>
</tbody>
</table>

### Distribution grid properties

These properties are specific to the distribution grid.

**Table 44. Distribution grid properties**

<table>
<thead>
<tr>
<th>Property name</th>
<th>Default value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rti.distgrid.gateway-receiver.bind-address</td>
<td></td>
<td>IPv4 address (dotted quad) for the interface for incoming connections from the ingest grid.</td>
</tr>
<tr>
<td>rti.gateway-receiver.hostname-for-senders and D.rti.gemfire.GatewayReceiver.HostNameForSenders</td>
<td></td>
<td>The hostname or IP address that remote ingest grid servers must use to establish connectivity. <strong>Note:</strong> These two properties must be set to the same value for RTI 2.1.</td>
</tr>
<tr>
<td>Property name</td>
<td>Default value</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------------------------------</td>
<td>---------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>rti.distgrid.gateway-receiver.port</td>
<td>55221</td>
<td>First TCP port for incoming connections from the ingest grid.</td>
</tr>
<tr>
<td>rti.distgrid.gateway-receiver.end-port</td>
<td>55300</td>
<td>Last TCP port for incoming connections from the ingest grid.</td>
</tr>
<tr>
<td>rti.distgrid.enrichers.geofenceid.contains.enable</td>
<td>false</td>
<td>Enable the enrichment of geofence data.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Type: boolean</td>
</tr>
<tr>
<td>rti.distgrid.enrichers.geofenceid.entryexit.enable</td>
<td>false</td>
<td>Enable the maintaining of state to know if a given event is an entry or exit event.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Type: boolean</td>
</tr>
<tr>
<td>rti.distgrid.enrichers.geofenceid.entryexit.timeout</td>
<td>300</td>
<td>The number of seconds to maintain state before the state becomes stale and should be discarded.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Type: integer</td>
</tr>
<tr>
<td>rti.distgrid.metrics.file.compress</td>
<td>true</td>
<td>Indicates whether or not metrics should be compressed.</td>
</tr>
</tbody>
</table>
### Chapter 14: Configuration Properties

<table>
<thead>
<tr>
<th>Property name</th>
<th>Default value</th>
<th>Description</th>
</tr>
</thead>
</table>
| rti.distgrid.metrics.<streamName>.statistics.enabled | false | Enable statistics gathering for a specific metrics stream. Available statistics: **Category**: SIMetricsExporter
Statistics for the export process. **Sub-category**: SIMetricsExporter-
<stream.name> Statistics for each stream.  
**skippedBuckets** Total number of metric buckets exported.  
**timeBucketsSent** Number of skipped buckets per export.  
The following additional properties are required:  
rti.gemfire.enable-time-statistics=true  
rti.gemfire.statistic-sampling-enabled=true  
rti.gemfire.statistic-archive-file=${rti.logs}/rti-distgrid-gemfire-stats.gfs  
rti.gemfire.statistic-sample-rate=5000  
See the GemFire documentation for information about these additional properties, which are set in the GemFire distributed system on startup: [http://gemfire.docs.pivotal.io/index.html](http://gemfire.docs.pivotal.io/index.html) |
| rti.distgrid.refdata.orgunit.allowExternalApiKey | false | Controls whether API keys can be provided when reference data for organization units is loaded. If this property is set to true, the administrator can set any API key for an OU. Otherwise, the key is generated by RTI. For security purposes, the default is false. |
| rti.distgrid.refdata.externalLocationOptinsLock.lease | 20000 | TimeMillis  
Controls how long to hold a lock taken when updating external LANDS opt-ins (by linking a geofence to a subscription). |
| rti.distgrid.refdata.externalLocationOptinsLock.wait | 30000 | TimeMillis  
Controls how long to wait for a lock that is being held by another user when updating external LANDS opt-ins (by linking a geofence to a subscription). |
<table>
<thead>
<tr>
<th>Property name</th>
<th>Default value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rti.distgrid.refdata.externalSubscribersOptins</td>
<td></td>
<td>Controls how long to hold a lock taken when updating external subscriber opt-ins via RDL and PM operations, such as IMSIs or MSISDNs being added to subscriptions with SSO messages. If you see the following message in the logs, increase the value of this property: &quot;Lease had expired. Concurrent updates may have happened, please review logs.&quot;</td>
</tr>
<tr>
<td>Lock.leaseTimeMillis</td>
<td>20000</td>
<td>Controls how long to hold a lock taken when updating external subscriber opt-ins via RDL and PM operations, such as IMSIs or MSISDNs being added to subscriptions with SSO messages.</td>
</tr>
<tr>
<td>rti.distgrid.refdata.externalSubscribersOptinsLoc</td>
<td></td>
<td>Controls how long to wait for a lock held by another user when updating external subscriber opt-ins via RDL and PM operations, such as IMSIs or MSISDNs being added to subscriptions with SSO messages. If you see the following message in the logs, increase the value of this property: &quot;Failed to lock External Subscriber optins for &lt;update&gt;/&lt;delete&gt;.&quot;</td>
</tr>
<tr>
<td>k.waitTimeMillis</td>
<td>30000</td>
<td>Controls how long to wait for a lock held by another user when updating external subscriber opt-ins via RDL and PM operations, such as IMSIs or MSISDNs being added to subscriptions with SSO messages. If you see the following message in the logs, increase the value of this property: &quot;Failed to lock External Subscriber optins for &lt;update&gt;/&lt;delete&gt;.&quot;</td>
</tr>
<tr>
<td>rti.distgrid.refdata.persistence</td>
<td></td>
<td>Controls the persistence of reference data and provisioning data. Controls the persistence of event data. The master region, which all other regions with an IMSI key are co-located with, is always persistent. This region is SUBSCRIBERS_Location</td>
</tr>
<tr>
<td>rti.distgrid.feed.pr.persistence</td>
<td>true false</td>
<td>Controls the persistence of reference data and provisioning data. Controls the persistence of event data. The master region, which all other regions with an IMSI key are co-located with, is always persistent. This region is SUBSCRIBERS_Location</td>
</tr>
<tr>
<td>rti.gateway.socket-buffer-size</td>
<td>4194304</td>
<td>Common to both grids, and must be set to the same value on both grids (not setting them the same could degrade performance).</td>
</tr>
<tr>
<td>rti.distgrid.geolocation.celltower.siteCoordinates</td>
<td>false</td>
<td>Whether to use cell center (centroid) location for geolocation (the default) or site location.</td>
</tr>
<tr>
<td>rti.indexes.geoSpatial.startupRebuildDelayMillis</td>
<td>1000</td>
<td>The frequency with which in-memory geospatial indexes associated with the cell tower data are rebuilt (in milliseconds).</td>
</tr>
</tbody>
</table>
## Chapter 14: Configuration Properties

<table>
<thead>
<tr>
<th>Property name</th>
<th>Default value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rti.distgrid.enrichers.cellTower.enabled</td>
<td>TRUE</td>
<td>Enables CellTower enrichment into the output payload by matching against cell ID properties</td>
</tr>
<tr>
<td>rti.distgrid.enrichers.cellTower.useSiteCoordinates</td>
<td>FALSE</td>
<td>Use the coordinate of the CellTower instead of the centroid</td>
</tr>
<tr>
<td>rti.distgrid.enrichers.device.enabled</td>
<td>TRUE</td>
<td>Enables enrichment of additional Device properties into output payload</td>
</tr>
<tr>
<td>rti.distgrid.enrichers.location.external.accuracy</td>
<td>0</td>
<td>EXPERIMENTAL: Enables enrichment of location based on externally provided data</td>
</tr>
<tr>
<td>rti.distgrid.enrichers.location.external.enabled</td>
<td>FALSE</td>
<td>EXPERIMENTAL: Enables enrichment of location based on externally provided data</td>
</tr>
<tr>
<td>rti.distgrid.enrichers.location.external.provider</td>
<td>ExternalLocation</td>
<td>EXPERIMENTAL: Enables enrichment of location based on externally provided data</td>
</tr>
<tr>
<td>rti.distgrid.enrichers.msisdn</td>
<td>FALSE</td>
<td>Enriches MSISDN field</td>
</tr>
<tr>
<td>rti.distgrid.enrichers.msisdnFromCallingDigits</td>
<td>FALSE</td>
<td>Enriches MSISDN field</td>
</tr>
<tr>
<td>rti.distgrid.enrichers.profile.enabled</td>
<td>TRUE</td>
<td>Enables SubscriberProfile enrichment by matching on IMSI</td>
</tr>
<tr>
<td>rti.distgrid.ess.batch.size</td>
<td>1</td>
<td>Maximum batch size to AMQP for ESS outputs</td>
</tr>
<tr>
<td>rti.distgrid.ess.batch.timeout</td>
<td>500</td>
<td>Maximum batch timeout to AMQP for ESS outputs</td>
</tr>
<tr>
<td>rti.distgrid.ess.compressPayload</td>
<td>FALSE</td>
<td>Should ESS output be compressed?</td>
</tr>
<tr>
<td>rti.distgrid.ess.encoding</td>
<td>UTF-8</td>
<td>Text encoding</td>
</tr>
<tr>
<td>rti.distgrid.ess.exchange</td>
<td>DECODER, STREAM</td>
<td>Default AMQP exchange</td>
</tr>
<tr>
<td>rti.distgrid.ess.groupedstream.timeoutSecs</td>
<td>600</td>
<td>Cleans expired ESS batch processors periodically</td>
</tr>
<tr>
<td>rti.distgrid.ess.message.persistent</td>
<td>FALSE</td>
<td>ESS messages persistent on AMQP?</td>
</tr>
<tr>
<td>rti.distgrid.ess.rabbit.queue.durable</td>
<td>TRUE</td>
<td>Controls whether the per-ESS RMQ queues are durable or not.</td>
</tr>
<tr>
<td>rti.distgrid.ess.routingKey</td>
<td>DECODER, STREAM .ALL</td>
<td>Default message routing key to ALL</td>
</tr>
</tbody>
</table>

[EMC Real-Time Intelligence (RTI) 3.1.0 Administrators Guide]
### Property name | Default value | Description
--- | --- | ---
rti.distgrid.format.barcelona.enableStatsInfo | TRUE | Adds stats info to output
rti.distgrid.format.barcelona.enableTagInfo | FALSE | Adds tag info field to output
rti.distgrid.format.barcelona.eventTimeAsUnixEpoch | TRUE | Specifies event time as UNIX epoch
rti.distgrid.format.barcelona.extTagInfo | TRUE | Adds extTag to output
rti.distgrid.format.barcelona.extendedTranFormat | TRUE | Adds extra fields to output
rti.distgrid.format.barcelona.geoCoordinatesMode | FALSE | Specifies geo-coordinates or cell id
rti.distgrid.format.barcelona.geoCoordinatesRadii90 | FALSE | 
rti.distgrid.format.location.eventTimeAsUnixEpoch | TRUE | Specifies event time as UNIX epoch
rti.distgrid.format.location.geoCoordinatesMode | FALSE | Specifies geo-coordinates or cell id
rti.distgrid.format.spel.format | protocolName, imsi, lac, cellTower, startTimeUTC, timeUTC, latitude,longitude, mccmnc, msisdn, imei, transactionTarget, firstLac, firstCellTower, eventType, eventStatus, protocolDetailMap, ingestTime, processedTime | Fields to output when using the csvFormat or SpEL output transformer
rti.distgrid.format.spel.preProcess | TRUE | Pre-processing converts "foo,bar" to "foo + ', ' + bar". If a more complex SpEL expression is needed, pre-processing can be disabled.
rti.distgrid.format.transaction.adrFormat | 0:callType: timeoutBits: bssMAPCause: dtapRRCause: dtapCCCause: dtapMMCause | Specifies ProtocolEvent Detail Map fields for ADR
<table>
<thead>
<tr>
<th>Property name</th>
<th>Default value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Property name</td>
<td>Default value</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>---------------</td>
<td>-------------</td>
</tr>
<tr>
<td>rti.distgrid.format.transaction.datacastS1MMEFormat</td>
<td>callType:xdrOption:application</td>
<td>Specifies ProtocolEvent Detail Map fields for S1MME</td>
</tr>
<tr>
<td></td>
<td>Protocol:csFallBackIndicator:transStatsType:transStatsStart</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Time:transStatsEndTime:transStatsCauses:transStatsStatus</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Bits:transStatsDirection:transProtocolId:ueambrMaxBitRateDownLink:ueambrMaxBitRateUpLink</td>
<td></td>
</tr>
<tr>
<td></td>
<td>eRABInfoCount:eRABInfoStatuseRABInfold:eRABInfoQci:eRABInfoPriorityLevel:eRABInfoPreemptionCapability:eRABInfoPreemptionVulnerable:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>eRABMbrDownLink:eRABMbrUpLink:eRABGbrDownLink:eRABGbrUpLink:eRABInfoCauseType:eRABInfoCauseValue</td>
<td></td>
</tr>
<tr>
<td>rti.distgrid.format.transaction.diameterFormat</td>
<td>visitedPlmnlD:user</td>
<td>Specifies ProtocolEvent Detail Map fields for Diameter</td>
</tr>
<tr>
<td></td>
<td>Name 3gppMeid:originRealm:destinationRealm:diameterApplicationId:releaseCause:callId:interfaceId</td>
<td></td>
</tr>
<tr>
<td>Property name</td>
<td>Default value</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------------------------------------------------</td>
<td>---------------</td>
<td>----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><code>rti.distgrid.format.transaction.diameterNativeFormat</code></td>
<td></td>
<td>Specifies ProtocolEvent Detail Map fields for Diameter</td>
</tr>
<tr>
<td><code>rti.distgrid.format.transaction.enableStatsInfo</code></td>
<td>TRUE</td>
<td>Adds stats info to output</td>
</tr>
<tr>
<td><code>rti.distgrid.format.transaction.enableTagInfo</code></td>
<td>FALSE</td>
<td>Adds tag info field to output</td>
</tr>
<tr>
<td><code>rti.distgrid.format.transaction.eventTimeAsUnixEpoch</code></td>
<td>TRUE</td>
<td>Specifies event time as UNIX epoch</td>
</tr>
<tr>
<td><code>rti.distgrid.format.transaction.extTagInfo</code></td>
<td>TRUE</td>
<td>Adds extTag to output</td>
</tr>
<tr>
<td><code>rti.distgrid.format.transaction.extendedTranFormat</code></td>
<td>TRUE</td>
<td>Adds extra fields to output</td>
</tr>
<tr>
<td><code>rti.distgrid.format.transaction.gbFormat</code></td>
<td><code>transactionType:0:timeoutBits:gmmsmCause:smsReleaseCause</code></td>
<td>Specifies ProtocolEvent Detail Map fields for GB</td>
</tr>
<tr>
<td><code>rti.distgrid.format.transaction.tdrFormat</code></td>
<td></td>
<td>Specifies ProtocolEvent Detail Map fields for TDR</td>
</tr>
<tr>
<td><code>rti.distgrid.format.transaction.geoCoordinatesMode</code></td>
<td>FALSE</td>
<td>Specifies geo- coordinates or cell id</td>
</tr>
<tr>
<td><code>rti.distgrid.format.transaction.geoCoordinatesRadii90</code></td>
<td>FALSE</td>
<td></td>
</tr>
<tr>
<td>Property name</td>
<td>Default value</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------------------------------------------</td>
<td>-------------------------------------------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>rti.distgrid.format.transaction.iucsFormat</td>
<td>transactionType: callType: timeoutBits: appProtocol: sccpCapCause: ranapCause: ccCause: mmCause</td>
<td>Specifies ProtocolEvent Detail Map fields for IUCS</td>
</tr>
<tr>
<td>rti.distgrid.format.transaction.iupsFormat</td>
<td>transactionType: callType: appProtocol: sccpCause: transactionRejectionCause</td>
<td>Specifies ProtocolEvent Detail Map fields for IUPS</td>
</tr>
</tbody>
</table>
## Chapter 14: Configuration Properties

<table>
<thead>
<tr>
<th>Property name</th>
<th>Default value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rti.distgrid.format.transaction.s1mmeFormat</td>
<td></td>
<td>Specifies ProtocolEvent Detail Map fields for S1MME.</td>
</tr>
<tr>
<td>rti.distgrid.format.transaction.tdrFormat</td>
<td></td>
<td>Specifies ProtocolEvent Detail Map fields for TDR.</td>
</tr>
<tr>
<td>rti.distgrid.lands.amqp.batchSize</td>
<td>2048</td>
<td>LANDS batch size to AMQP.</td>
</tr>
<tr>
<td>rti.distgrid.lands.amqp.enabled</td>
<td>FALSE</td>
<td>Enables LANDS output to AMQP.</td>
</tr>
<tr>
<td>rti.distgrid.lands.amqp.encoding</td>
<td>UTF-8</td>
<td>Text encoding.</td>
</tr>
<tr>
<td>rti.distgrid.lands.amqp.exchange</td>
<td>DECODER.STREAM</td>
<td>AMQP Exchange target for LANDS.</td>
</tr>
<tr>
<td>rti.distgrid.lands.amqp.message.persistent</td>
<td>FALSE</td>
<td>Is message persistent on broker?</td>
</tr>
<tr>
<td>rti.distgrid.lands.amqp.routingKey</td>
<td>DECODER,STREAM,AL.L.LANDS</td>
<td>AMQP routing key.</td>
</tr>
<tr>
<td>rti.distgrid.lands.amqp.timeout</td>
<td>4000</td>
<td>LANDS batch timeout.</td>
</tr>
<tr>
<td>rti.distgrid.lands.file.batchSize</td>
<td>8192</td>
<td>Batch size.</td>
</tr>
<tr>
<td>rti.distgrid.lands.file.compress</td>
<td>FALSE</td>
<td>Compress text.</td>
</tr>
<tr>
<td>rti.distgrid.lands.file.compressionDelay</td>
<td>1000</td>
<td>Delay after which output files are compressed.</td>
</tr>
<tr>
<td>Property name</td>
<td>Default value</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------------------------</td>
<td>---------------------------------------------------------</td>
<td>---------------------------------------------</td>
</tr>
<tr>
<td>rti.distgrid.lands.file.dateFormat</td>
<td>yyyyMMdd'T'HHmmssSS</td>
<td>Date format of file name</td>
</tr>
<tr>
<td>rti.distgrid.lands.file.enabled</td>
<td>FALSE</td>
<td>Enables output to file</td>
</tr>
<tr>
<td>rti.distgrid.lands.file.path</td>
<td>${rti.data}/lands</td>
<td>Specifies path to output directory</td>
</tr>
<tr>
<td>rti.distgrid.lands.file.timeout</td>
<td>4000</td>
<td>Batch timeout</td>
</tr>
<tr>
<td>rti.distgrid.lands.format.enabled</td>
<td>FALSE</td>
<td>Master switch for LANDS output</td>
</tr>
<tr>
<td>rti.distgrid.lands.format</td>
<td>location</td>
<td>LANDS output format type</td>
</tr>
<tr>
<td>rti.distgrid.lands.location.disabled</td>
<td>FALSE</td>
<td>Enables location check in LANDS opt-in</td>
</tr>
<tr>
<td>rti.distgrid.lands.location.keyoptimization</td>
<td>FALSE</td>
<td>Enable opt-in mode is whitelist or blacklist</td>
</tr>
<tr>
<td>rti.distgrid.lands.subscriber.disabled</td>
<td>FALSE</td>
<td>Enables subscriber check in LANDS opt-in</td>
</tr>
<tr>
<td>rti.distgrid.lands.subscriber.keyoptimization</td>
<td>FALSE</td>
<td>Enable opt-in mode is whitelist or blacklist</td>
</tr>
<tr>
<td>rti.distgrid.lors.analytics.enabled</td>
<td>TRUE</td>
<td>Enables RTI Analytics</td>
</tr>
<tr>
<td>rti.distgrid.lors.metrics.enabled</td>
<td>TRUE</td>
<td>Enables RTI Metrics</td>
</tr>
<tr>
<td>rti.distgrid.lors.amqp.batchSize</td>
<td>2048</td>
<td>LORS batch size to AMQP</td>
</tr>
<tr>
<td>rti.distgrid.lors.amqp.enabled</td>
<td>FALSE</td>
<td>Enables LORS output to AMQP</td>
</tr>
<tr>
<td>rti.distgrid.lors.amqp.encoding</td>
<td>UTF-8</td>
<td>Text encoding</td>
</tr>
<tr>
<td>rti.distgrid.lors.amqp.exchange</td>
<td>DECODER.STREAM</td>
<td>AMQP Exchange target for LORS</td>
</tr>
<tr>
<td>rti.distgrid.lors.amqp.message.persistent</td>
<td>FALSE</td>
<td>Is message persistent on broker?</td>
</tr>
<tr>
<td>rti.distgrid.lors.amqp.routingKey</td>
<td>DECODER.STREAMAL.L.LORS</td>
<td>AMQP routing key</td>
</tr>
<tr>
<td>rti.distgrid.lors.amqp.timeout</td>
<td>4000</td>
<td>LORS batch timeout</td>
</tr>
<tr>
<td>rti.distgrid.lors.file.batchSize</td>
<td>8192</td>
<td>Batch size</td>
</tr>
<tr>
<td>rti.distgrid.lors.file.compress</td>
<td>FALSE</td>
<td>Compresses text</td>
</tr>
<tr>
<td>rti.distgrid.lors.file.compressionDelay</td>
<td>1000</td>
<td>Delay after which output files are compressed</td>
</tr>
<tr>
<td>rti.distgrid.lors.file.dateFormat</td>
<td>yyyyMMdd'T'HHmmssSS</td>
<td>Date format of file name</td>
</tr>
<tr>
<td>rti.distgrid.lors.file.enabled</td>
<td>TRUE</td>
<td>Enables output to file</td>
</tr>
</tbody>
</table>
## Chapter 14: Configuration Properties

<table>
<thead>
<tr>
<th>Property name</th>
<th>Default value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rti.distgrid.lors.file.path</td>
<td><code>${rti.data}/lors</code></td>
<td>Specifies path to output directory</td>
</tr>
<tr>
<td>rti.distgrid.lors.file.timeout</td>
<td>4000</td>
<td>Batch timeout</td>
</tr>
<tr>
<td>rti.distgrid.lors.jdbc.abandonWhenPercentageFull</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>rti.distgrid.lors.jdbc.accessToUnderlyingConnectionAllowed</td>
<td></td>
<td></td>
</tr>
<tr>
<td>rti.distgrid.lors.jdbc.alternateUsernameAllowed</td>
<td>FALSE</td>
<td></td>
</tr>
<tr>
<td>rti.distgrid.lors.jdbc.batchSize</td>
<td>2048</td>
<td></td>
</tr>
<tr>
<td>rti.distgrid.lors.jdbc.commitOnReturn</td>
<td>FALSE</td>
<td></td>
</tr>
<tr>
<td>rti.distgrid.lors.jdbc.connectionProperties</td>
<td></td>
<td></td>
</tr>
<tr>
<td>rti.distgrid.lors.jdbc.defaultAutoCommit</td>
<td>FALSE</td>
<td></td>
</tr>
<tr>
<td>rti.distgrid.lors.jdbc.defaultCatalog</td>
<td>APP</td>
<td></td>
</tr>
<tr>
<td>rti.distgrid.lors.jdbc.defaultReadOnly</td>
<td>FALSE</td>
<td></td>
</tr>
<tr>
<td>rti.distgrid.lors.jdbc.defaultTransactionIsolation</td>
<td>-1</td>
<td></td>
</tr>
<tr>
<td>rti.distgrid.lors.jdbc.driverClassName</td>
<td></td>
<td></td>
</tr>
<tr>
<td>rti.distgrid.lors.jdbc.enabled</td>
<td>FALSE</td>
<td></td>
</tr>
<tr>
<td>rti.distgrid.lors.jdbc.fairQueue</td>
<td>TRUE</td>
<td></td>
</tr>
<tr>
<td>rti.distgrid.lors.jdbc.ignoreExceptionOnPreLoad</td>
<td>FALSE</td>
<td></td>
</tr>
<tr>
<td>rti.distgrid.lors.jdbc.initSQL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>rti.distgrid.lors.jdbc.insertStatement</td>
<td>INSERT INTO RTI.RTI_LANDING</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(payload, feed) VALUES (?, ?)</td>
<td></td>
</tr>
<tr>
<td>rti.distgrid.lors.jdbc.jmxEnabled</td>
<td>TRUE</td>
<td></td>
</tr>
<tr>
<td>rti.distgrid.lors.jdbc.logAbandoned</td>
<td>FALSE</td>
<td></td>
</tr>
<tr>
<td>rti.distgrid.lors.jdbc.logValidationErrors</td>
<td>FALSE</td>
<td></td>
</tr>
<tr>
<td>rti.distgrid.lors.jdbc.maxActive</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>rti.distgrid.lors.jdbc.maxAge</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>rti.distgrid.lors.jdbc.maxIdle</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>rti.distgrid.lors.jdbc.maxWait</td>
<td>30000</td>
<td></td>
</tr>
<tr>
<td>rti.distgrid.lors.jdbc.minEvictableIdleTimeMillis</td>
<td>60000</td>
<td></td>
</tr>
<tr>
<td>rti.distgrid.lors.jdbc.minIdle</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>Property name</td>
<td>Default value</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------------------------------</td>
<td>---------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>rti.distgrid.lors.jdbc.numTestsPerEvictionRun</td>
<td>-1</td>
<td></td>
</tr>
<tr>
<td>rti.distgrid.lors.jdbc.password</td>
<td>guest</td>
<td></td>
</tr>
<tr>
<td>rti.distgrid.lors.jdbc.propagateInterruptState</td>
<td>FALSE</td>
<td></td>
</tr>
<tr>
<td>rti.distgrid.lors.jdbc.removeAbandoned</td>
<td>FALSE</td>
<td></td>
</tr>
<tr>
<td>rti.distgrid.lors.jdbc.removeAbandonedTimeout</td>
<td>60</td>
<td></td>
</tr>
<tr>
<td>rti.distgrid.lors.jdbc.rollbackOnReturn</td>
<td>FALSE</td>
<td></td>
</tr>
<tr>
<td>rti.distgrid.lors.jdbc.suspectTimeout</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>rti.distgrid.lors.jdbc.testOnBorrow</td>
<td>FALSE</td>
<td></td>
</tr>
<tr>
<td>rti.distgrid.lors.jdbc.testOnConnect</td>
<td>FALSE</td>
<td></td>
</tr>
<tr>
<td>rti.distgrid.lors.jdbc.testOnReturn</td>
<td>FALSE</td>
<td></td>
</tr>
<tr>
<td>rti.distgrid.lors.jdbc.testWhileIdle</td>
<td>FALSE</td>
<td></td>
</tr>
<tr>
<td>rti.distgrid.lors.jdbc.timeBetweenEvictionRuns</td>
<td>5000</td>
<td></td>
</tr>
<tr>
<td>rti.distgrid.lors.jdbc.timeout</td>
<td>4000</td>
<td></td>
</tr>
<tr>
<td>rti.distgrid.lors.jdbc.url</td>
<td></td>
<td></td>
</tr>
<tr>
<td>rti.distgrid.lors.jdbc.useDisposableConnectionFactory</td>
<td>TRUE</td>
<td>LORS output format type</td>
</tr>
<tr>
<td>rti.distgrid.lors.jdbc.useEquals</td>
<td>TRUE</td>
<td></td>
</tr>
<tr>
<td>rti.distgrid.lors.jdbc.useLock</td>
<td>FALSE</td>
<td></td>
</tr>
<tr>
<td>rti.distgrid.lors.jdbc.username</td>
<td>guest</td>
<td></td>
</tr>
<tr>
<td>rti.distgrid.lors.jdbc.validationInterval</td>
<td>30000</td>
<td></td>
</tr>
<tr>
<td>rti.distgrid.lors.jdbc.validationQuery</td>
<td>SELECT 1</td>
<td></td>
</tr>
<tr>
<td>rti.distgrid.lors.jdbc.validationQueryTimeout</td>
<td>-1</td>
<td></td>
</tr>
<tr>
<td>rti.distgrid.lors.format</td>
<td>transaction</td>
<td></td>
</tr>
<tr>
<td>rti.distgrid.lors.location.disabled</td>
<td>FALSE</td>
<td>Disables location check in LORS opt-in</td>
</tr>
<tr>
<td>rti.distgrid.lors.location.keyoptimization</td>
<td>FALSE</td>
<td></td>
</tr>
<tr>
<td>rti.distgrid.lors.mode</td>
<td>whitelist</td>
<td>Opt-in mode is whitelist or blacklist</td>
</tr>
<tr>
<td>rti.distgrid.lors.stateful.enabled</td>
<td>FALSE</td>
<td></td>
</tr>
<tr>
<td>rti.distgrid.lors.subscriber.disabled</td>
<td>FALSE</td>
<td>Disables subscriber check in LORS opt-in</td>
</tr>
<tr>
<td>rti.distgrid.lors.subscriber.keyoptimization</td>
<td>FALSE</td>
<td></td>
</tr>
<tr>
<td>rti.distgrid.outputs.ignoreFailures</td>
<td>TRUE</td>
<td>Ignores errors if events don't match a known output route</td>
</tr>
<tr>
<td>Property name</td>
<td>Default value</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------------</td>
<td>---------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>rti.logger.backlog</td>
<td>65536</td>
<td>Pipeline logging is asynchronous, specifies thread pool backlog capacity</td>
</tr>
<tr>
<td>rti.logger.poolsize</td>
<td>2</td>
<td>Pipeline logging is asynchronous, specifies thread count</td>
</tr>
<tr>
<td>rti.output.backlog</td>
<td>65536</td>
<td>Output IO is asynchronous, specifies thread pool backlog capacity</td>
</tr>
<tr>
<td>rti.output.poolsize</td>
<td>32</td>
<td>Output IO is asynchronous, specifies thread count</td>
</tr>
<tr>
<td>rti.trace.channels.pattern</td>
<td>*</td>
<td>Message pattern match for trace logging</td>
</tr>
<tr>
<td>rti.trace.channels</td>
<td>TRUE</td>
<td>Enables pipeline logging to trace log</td>
</tr>
<tr>
<td>rti.distgrid.rabbit.catchAllExchange.durable</td>
<td>TRUE</td>
<td>Controls whether the catchAll RMQ Exchange is durable.</td>
</tr>
<tr>
<td>rti.distgrid.rabbit.catchAllQueue.durable</td>
<td>TRUE</td>
<td>Controls whether the cacheAll RMQ Queue is durable.</td>
</tr>
<tr>
<td>rti.distgrid.ess.rabbit.queue.durable</td>
<td>TRUE</td>
<td>Controls whether the per-ESS RMQ queues are durable or not.</td>
</tr>
<tr>
<td>rti.distgrid.analytics.validation.enabled</td>
<td>FALSE</td>
<td>This property needs to be enabled by setting the value to 'true' for every distribution grid in the system. When analytics validation mode has been enabled (by enabling this property), the distribution grid sends more analytics information to Syphon by way of amqp than it can use. When this option is enabled, the values for options rti.distgrid.syphon.amqp.exchange and rti.distgrid.syphon.amqp.routingKey also need to be changed.</td>
</tr>
<tr>
<td>Property name</td>
<td>Default value</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------------------------------</td>
<td>----------------</td>
<td>-------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>rti.distgrid.syphon.amqp.exchange</td>
<td>DECODER.STREAM</td>
<td>AMQP exchange target for SYPHON. When the rti.distgrid.analytics.validation.enabled option is enabled, the values for this property and rti.distgrid.syphon.amqp.routingKey property also need to be changed.</td>
</tr>
<tr>
<td>rti.distgrid.syphon.amqp.routingKey</td>
<td>DECODER.STREAM. ALL.SYPHON</td>
<td>Distribution grid AMQP routing key. When the rti.distgrid.analytics.validation.enabled option is enabled, the values for the rti.distgrid.syphon.amqp.exchange property and this property also need to be changed.</td>
</tr>
<tr>
<td>rti.transformers.anonymizeAnyField.enabled</td>
<td>false</td>
<td>Enables data anonymization. For more information refer to Anonymizing reference data</td>
</tr>
<tr>
<td>rti.transformers.anonymizeAnyField.fields</td>
<td></td>
<td>Used for anonymizing data. After enabling data anonymization, use a CSV list of the fields to be anonymized in this property. For more information refer to Anonymizing reference data</td>
</tr>
</tbody>
</table>

**Provisioning Manager and RabbitMQ properties**

These properties are specific to the Provisioning Manager and RabbitMQ.

Table 45. Provisioning manager and RabbitMQ properties

<table>
<thead>
<tr>
<th>Property name</th>
<th>Default value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rti.provisioning.msg.maxMessageBytes</td>
<td>10485760</td>
<td>Maximum message size.</td>
</tr>
<tr>
<td>rti.provisioning.msg.titleTranslationEnabled</td>
<td>true</td>
<td>Allow or disallow translation to be set as a message flag.</td>
</tr>
<tr>
<td>rti.provisioning.msg.lineSeparator</td>
<td>\n</td>
<td>Line separator to be used in messages.</td>
</tr>
</tbody>
</table>
### Chapter 14: Configuration Properties

<table>
<thead>
<tr>
<th>Property name</th>
<th>Default value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rti.provisioning.msg.includeContextInReplyMessage</td>
<td>false</td>
<td>Return extended information in the reply.</td>
</tr>
<tr>
<td>rti.provisioning.msg.gzipControlMessages</td>
<td>false</td>
<td>Compress (gzip) control messages.</td>
</tr>
<tr>
<td>rti.provisioning.msg.controlMessageTtlMillis</td>
<td>-1</td>
<td>Time to live on RabbitMQ queue for send control message. -1 means no upper limit.</td>
</tr>
<tr>
<td>rti.provisioning.replyToEnabled</td>
<td>true</td>
<td>Enables reading the AMQP replyTo header and sending the reply there instead of using the default reply queue.</td>
</tr>
<tr>
<td>rti.provisioning.batch.size</td>
<td>5000</td>
<td>Number of entries to process at a time for messages that contain multiple rows of input.</td>
</tr>
<tr>
<td>rti.provisioning.maxQueueMessages</td>
<td>-1</td>
<td>Maximum number of messages in the RabbitMQ queues set up for an OU by the Provisioning Manager. -1 means no upper limit.</td>
</tr>
<tr>
<td>rti.provisioning.rabbit.ou.poolSize</td>
<td>1 to 4</td>
<td>Number of threads available per OU for executing provisioning requests. The default is 1 thread with a bursting capacity up to 4.</td>
</tr>
<tr>
<td>rti.provisioning.rabbit.ou.queueCapacity</td>
<td>20</td>
<td>Number of provisioning requests to hold in memory waiting to be executed.</td>
</tr>
<tr>
<td>rti.provisioning.streams.exchange</td>
<td>DECODER.STREAM</td>
<td>Name of the exchange that subscription streams are linked to.</td>
</tr>
<tr>
<td>rti.provisioning.streams.streamQueueTTL</td>
<td>60000</td>
<td>Time to live on the queue, in milliseconds, for a subscription set up by the Provisioning Manager.</td>
</tr>
<tr>
<td>rti.rabbit.addresses</td>
<td>localhost</td>
<td>Comma-separated list of hostnames or host and port combinations for connections to rabbitMQ. For example: host1, host2:12345</td>
</tr>
<tr>
<td>rti.rabbit.channelCacheSize</td>
<td>50</td>
<td>Number of cached RabbitMQ channels. Increase this value if the number of OUs exceeds 50.</td>
</tr>
</tbody>
</table>
### Event format properties

These properties influence the content of the output for event streams. Set these properties in the distribution grid. See also [Output Format for Event Streams](#).

<table>
<thead>
<tr>
<th>Property name</th>
<th>Default value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rti.distgrid.amqp-batch.num-events</td>
<td>10000</td>
<td>Maximum number of events to collect before sending an AMQP events message.</td>
</tr>
<tr>
<td>rti.distgrid.amqp-batch.num-seconds-timeout</td>
<td>1</td>
<td>Maximum number of seconds to collect events before sending an AMQP events message.</td>
</tr>
<tr>
<td>rti.subscriber.events.batch-time-seconds</td>
<td>10</td>
<td>Maximum number of seconds to collect events before writing them out to a file.</td>
</tr>
<tr>
<td>Property name</td>
<td>Default value</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------------------------</td>
<td>---------------</td>
<td>-------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>rti.subscriber.events.batch-size</td>
<td>10000</td>
<td>Maximum number of events to collect before writing them to file (by default, 10000 events per batch).</td>
</tr>
<tr>
<td>rti.distgrid.format.csvField.format</td>
<td>imsi,imei</td>
<td>List of fields to include in a configured output format named csvField</td>
</tr>
<tr>
<td>rti.distgrid.format.location.dir</td>
<td>${rti.data}/locations</td>
<td>Folders for storing CSV files for location and transaction events.</td>
</tr>
<tr>
<td>rti.distgrid.format.transaction.dir</td>
<td>${rti.data}/transactions</td>
<td>Folders for storing CSV files for location and transaction events.</td>
</tr>
<tr>
<td>rti.distgrid.lands.format</td>
<td>location</td>
<td>Name of output format for the location events stream. csvField and configured output format names are valid.</td>
</tr>
<tr>
<td>rti.distgrid.lors.format</td>
<td>transaction</td>
<td>Name of output format for the transaction events stream. csvField and configured output format names are valid.</td>
</tr>
<tr>
<td>rti.distgrid.format.transaction.eventTimeAsUnixEpoch</td>
<td>true</td>
<td>Output event times in seconds (true) or milliseconds (false). Separate properties for transaction and location events.</td>
</tr>
<tr>
<td>rti.distgrid.format.location.eventTimeAsUnixEpoch</td>
<td>true</td>
<td>Output event times in seconds (true) or milliseconds (false). Separate properties for transaction and location events.</td>
</tr>
<tr>
<td>rti.distgrid.format.transaction.geoCoordinatesMode</td>
<td>false</td>
<td>Output the LAC/cell ID (false) or latitude/longitude (true). Separate properties for transaction and location events.</td>
</tr>
<tr>
<td>rti.distgrid.format.location.geoCoordinatesMode</td>
<td>false</td>
<td>Output the LAC/cell ID (false) or latitude/longitude (true). Separate properties for transaction and location events.</td>
</tr>
<tr>
<td>rti.distgrid.format.transaction.geoCoordinatesRadii90</td>
<td>false</td>
<td>Add an extra 90% radii field to the output (if true). This field is inserted as field 12.</td>
</tr>
<tr>
<td>rti.distgrid.format.transaction.decodeStatusBits</td>
<td>false</td>
<td>Decode event status details into field 14 of the output (if true).</td>
</tr>
</tbody>
</table>

**Note:** Field 14 is deprecated. Complete the instructions in Configuring Output Streams to configure output streams.
<table>
<thead>
<tr>
<th>Property name</th>
<th>Default value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rti.distgrid.format.transaction.enableStatsInfo</td>
<td>true</td>
<td>Adds the three timestamp fields to the output: 15 through 17 (if true).</td>
</tr>
<tr>
<td>rti.distgrid.format.transaction.enableTagInfo</td>
<td>false</td>
<td>Puts tag information into field 18 (if true).</td>
</tr>
<tr>
<td>rti.distgrid.format.transaction.extendedTranFormat</td>
<td>true</td>
<td>Adds protocol-specific data to field 14 (if true).</td>
</tr>
<tr>
<td>rti.distgrid.format.transaction.extTagInfo</td>
<td>true</td>
<td>Disables field 13, the Tag field. (Field 14 becomes field 13.)</td>
</tr>
</tbody>
</table>
RDL properties

These properties are specific to the RDL. The properties in the following table are set in this file:

```
emc_rti_<version>/code/admin/refdataloader/src/main/resources/refdataloader- defaults.properties
```

You can modify the values for these properties.

**Table 47. RDL properties**

<table>
<thead>
<tr>
<th>Property name</th>
<th>Default value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>celltower.batch.size</td>
<td>5000</td>
<td>Batch size of CSV reference data records loaded into the ingest grid. Modify the batch size based on the size of the record (number of fields) and the system resources available to RDL. If the records contain only a few fields and the RDL system has powerful resources (or its own hardware), consider increasing the batch size to improve I/O performance on the grid. A smaller batch size may be appropriate for data types loaded from smaller files. The listed default values typically provide good performance.</td>
</tr>
<tr>
<td>devices.batch.size</td>
<td>1000</td>
<td></td>
</tr>
<tr>
<td>geofence.batch.size</td>
<td>1000</td>
<td></td>
</tr>
<tr>
<td>locationOptin.batch.size</td>
<td>1000</td>
<td></td>
</tr>
<tr>
<td>orgunit.batch.size</td>
<td>1000</td>
<td></td>
</tr>
<tr>
<td>subOptin.batch.size</td>
<td>1000</td>
<td></td>
</tr>
<tr>
<td>subProfile.batch.size</td>
<td>1000</td>
<td></td>
</tr>
<tr>
<td>trace.batch.size</td>
<td>1000</td>
<td></td>
</tr>
<tr>
<td>linkgeosub.batch.size</td>
<td>1000</td>
<td></td>
</tr>
<tr>
<td>subscriptions.batch.size</td>
<td>1000</td>
<td></td>
</tr>
<tr>
<td>subscriberMapping.batch.size</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>max.lengthsubscriber.group.name</td>
<td>8</td>
<td>The maximum length of the subscriber group name.</td>
</tr>
<tr>
<td>rti-loader.prompt</td>
<td>rti&gt;</td>
<td>Prompt for the RDL command-line interface.</td>
</tr>
<tr>
<td>subscriber.trace.anonymize</td>
<td>true</td>
<td>Enables you to override the global rti.anonymizer.noHashIMSI system setting to hash the IMSI for the Trace Subscriber data type record. If false, then the IMSI is not hashed even if the global system setting specifies to hash the IMSIs. If true, then the global system setting takes precedence. IMSIs are hashed or not, depending on that setting.</td>
</tr>
</tbody>
</table>
REST API client properties

The following properties are configured in the /etc/rest-client/application.properties file.

Table 48. REST API client properties

<table>
<thead>
<tr>
<th>Property name</th>
<th>Default value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>server.contextPath</td>
<td>/rti-t</td>
<td>This property is not configurable. The default URI to start accessing the rest client API is as follows: http://&lt;rest app hostname&gt;:&lt;rest app port&gt;/rti-t</td>
</tr>
<tr>
<td>server.port</td>
<td>8080</td>
<td>Server port number.</td>
</tr>
<tr>
<td>server.address</td>
<td>localhost</td>
<td>Server IP address for the REST API application.</td>
</tr>
<tr>
<td>server.sessionTimeout</td>
<td>30</td>
<td>Server session timeout in seconds, intended for use by applications that invoke the REST API client. This timeout value represents the length of idle time the client allows before closing the session.</td>
</tr>
</tbody>
</table>

HDFS properties

The following HDFS property can be configured.

Table 49. HDFS properties

<table>
<thead>
<tr>
<th>Property name</th>
<th>Default value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rti.hdfs.transport.encode</td>
<td>false</td>
<td>Indicates whether to convert raw binary data to hexadecimal format. By default, RTI writes data in raw binary format.</td>
</tr>
</tbody>
</table>
This chapter presents the following topics:

- Referencing RDL commands ................................................................. 273
- buildSpatialIndex command .............................................................. 273
- create command .............................................................................. 274
- delete command ............................................................................... 275
- fetchSubscriberMapping command ................................................. 276
- rebuildOptins command .................................................................. 276
- subscriptionState command .............................................................. 277
- update command ............................................................................... 277
RDL commands

This section contains syntax descriptions for the RDL commands. Type `help` on the RDL command line for similar information.

Several generic commands can be used on both the `rtish` command line and the `rdl` command line. The following table summarizes the commands that are specific to the loader. See `RTISH commands` for the generic RTI commands.

### Table 50. RDL commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>buildspatialIndex</code></td>
<td>Forces the cell tower reference data to be reindexed. By default, cell tower data that flows into the distribution grid is not indexed automatically. You can use this command to index it.</td>
</tr>
<tr>
<td><code>create</code></td>
<td>Loads data from the specified CSV file for the specified reference data type.</td>
</tr>
<tr>
<td><code>delete</code></td>
<td>Deletes entries based on key values in the specified file or remove the whole region for a specified reference data type.</td>
</tr>
<tr>
<td><code>fetchSubscriberMapping</code></td>
<td>Generates the subscriber profile mapping file, which tells you the actual IMSI values that map to downstream hashed IMSI values in the RTI system. IMSI values are hashed to anonymize them.</td>
</tr>
<tr>
<td><code>rebuildOptIns</code></td>
<td>Cleans up and rebuilds the external location and subscriber opt-in data in the distribution grid. When you set up a subscription, it has an expiration date. This command checks the internal indexes and cleans out any expired subscriptions. This cleanup optimizes runtime performance and should be run on a daily basis.</td>
</tr>
<tr>
<td><code>subscriptionState</code></td>
<td>Turns specified event subscriptions on or off. (A subscription in this context is the mapping of IMSI values and geofences.)</td>
</tr>
<tr>
<td><code>update</code></td>
<td>Inserts or updates entries from the specified file for the specified reference data type.</td>
</tr>
</tbody>
</table>

**Important:** In production deployments of RTI, do not use `create` or `update` commands on the following data types:
- Subscriptions
- Geofences
- Linkgeofences
- Subscriber opt-ins with External opt-in type
- LANDS opt-ins with External opt-in type

Use these commands only to bootstrap the system.

### buildSpatialIndex command

**Description**

Force the cell tower reference data to be reindexed. By default, cell tower data that flows into the distribution grid does not get indexed automatically. You can use this command to index it.
Chapter 15: Referencing RDL Commands

Syntax
buildspatialIndex --freeze <true/false>

create command

Description
Load data from the specified CSV file for the specified reference data type.

Syntax
create --file <filename> --datatype <type> --optintype <type> --tracetype <type> --translate <value>

Arguments
Table 51. RDL create command arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Required/optional</th>
<th>Values</th>
<th>Default (if specified)</th>
<th>Default (if not specified)</th>
</tr>
</thead>
<tbody>
<tr>
<td>--file</td>
<td>Required</td>
<td>Path to a CSV file</td>
<td>No default value</td>
<td>No default value</td>
</tr>
<tr>
<td>--datatype</td>
<td>Required</td>
<td>CellTower, Device, Geofence, Linkgeofence, LocationOptin, Orgunit, SubscriberOptin, SubscriberProfile, Subscriptions, Trace</td>
<td>No default value</td>
<td>No default value</td>
</tr>
<tr>
<td>--optintype</td>
<td>Optional</td>
<td>Blacklist, DR, External, Global, LANDS, LORS, Opt-in type applies only if the data type is LocationOptin or SubscriberOptin</td>
<td>No default value.</td>
<td></td>
</tr>
<tr>
<td>--tracetype</td>
<td>Optional</td>
<td>Device, Location, Subscriber, Trace type applies only if the data type is Trace.</td>
<td>No default value</td>
<td>No default value</td>
</tr>
</tbody>
</table>
### create Examples

```bash
create --file refdata/location_optin.csv --datatype LocationOptin --optintype Global
create --file refdata/subscriber_optin.csv --datatype SubscriberOptin --optintype Global
create --file refdata/cell_tower.csv --datatype CellTower
```

**Important:** In production deployments of RTI, do not use `create` or `update` commands on the following data types:
- Subscriptions
- Geofences
- Linkgeofences
- Subscriber opt-ins with External opt-in type
- LANDS opt-ins with External opt-in type

Use these commands only to bootstrap the system.

### delete command

**Description**
Delete entries based on key values in the specified file or remove the whole region for a specified reference data type.

**Syntax**
```bash
delete --file <filename> --datatype <type> --optintype <type> --tracetype <type> --translate <value> --force <value>
```

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Required/optional</th>
<th>Values</th>
<th>Default (if specified)</th>
<th>Default (if not specified)</th>
</tr>
</thead>
<tbody>
<tr>
<td>--file</td>
<td>Optional</td>
<td>Path to a CSV file</td>
<td>No default value</td>
<td>No default value</td>
</tr>
</tbody>
</table>
## Chapter 15: Referencing RDL Commands

### Argument Table

<table>
<thead>
<tr>
<th>Argument</th>
<th>Required/optional</th>
<th>Values</th>
<th>Default (if specified)</th>
<th>Default (if not specified)</th>
</tr>
</thead>
<tbody>
<tr>
<td>--datatype</td>
<td>Required</td>
<td>CellTower, Device, Geofence, Linkgeofence, LocationOptin, Orgunit, SubscriberOptin, SubscriberProfile, Subscriptions, Trace</td>
<td>No default value</td>
<td>No default value</td>
</tr>
<tr>
<td>--optintype</td>
<td>Optional</td>
<td>Blacklist, DR, External, Global, LANDS, LORS</td>
<td>No default value</td>
<td>No default value</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Opt-in type applies only if the data type is LocationOptin or SubscriberOptin.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>--tracetype</td>
<td>Optional</td>
<td>Device, Location, Subscriber Trace type applies only if the data type is Trace.</td>
<td>No default value</td>
<td>No default value</td>
</tr>
<tr>
<td>--translate</td>
<td>Optional</td>
<td>0 or 1, false or 'false', true, no or yes Translate applies only if the data type is SubscriberOptin.</td>
<td>'false'</td>
<td>No default value</td>
</tr>
<tr>
<td>--force</td>
<td>Optional</td>
<td>0 or 1, false or 'true', true, no or yes</td>
<td>'true'</td>
<td>'true'</td>
</tr>
</tbody>
</table>

### delete Example

```
delete --datatype CellTower --force true
```

### fetchSubscriberMapping command

#### Description
Generate the subscriber profile mapping file, which will tell you the actual IMSI values that map to downstream hashed IMSI values in the RTI system. IMSI values are hashed to anonymize them.

#### Syntax
```
fetchSubscriberMapping --optintype <type> --inputfile <file> --outputfile <file> --translate <value>
```

### rebuildOptins command

#### Description
Clean and rebuild the external location and subscriber opt-in data in the distribution grid. When you set up a subscription, it has an expiration date. This command checks the internal indexes and cleans out any expired subscriptions. This cleaning optimizes runtime performance and should be run on a daily basis.
This command applies to the external opt-in data in the distribution grid only and should not be run against other components.

**Syntax**

rebuildOptIns --type <type> --simulate <value>

**Arguments**

Table 53. RDL rebuildOptIns command arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Required/optional</th>
<th>Values</th>
<th>Default (if specified)</th>
<th>Default (if not specified)</th>
</tr>
</thead>
<tbody>
<tr>
<td>--type</td>
<td>Required</td>
<td>ExOptInRebuildAll, LocationOptInRebuild, SubscriptionOptInRebuild</td>
<td>No default value</td>
<td>No default value</td>
</tr>
<tr>
<td>--simulate</td>
<td>Optional</td>
<td>0 or 1, false or true, no or yes</td>
<td>'true'</td>
<td>'false'</td>
</tr>
</tbody>
</table>

**subscriptionState command**

**Description**

Turn specified event subscriptions on or off. (A subscription in this context is the mapping of IMSI values and geofences.)

**Syntax**

subscriptionState --file <filename>

**update command**

**Description**

Insert or update entries from the specified file for the specified reference data type.

**Important:** In production deployments of RTI, do not use create or update commands on the following data types:
- Subscriptions
- Geofences
- Linkgeofences
- Subscriber opt-ins with External opt-in type
- LANDS opt-ins with External opt-in type

Use these commands only to bootstrap the system.

**Syntax**

update --file <filename> --datatype <type> --optintype <type> --tracetype <type> --translate <value>
Arguments

Table 54. RDL update command arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Required/optional</th>
<th>Values</th>
<th>Default (if specified)</th>
<th>Default (if not specified)</th>
</tr>
</thead>
<tbody>
<tr>
<td>--file</td>
<td>Required</td>
<td>Path to a CSV file</td>
<td>No default value</td>
<td>No default value</td>
</tr>
<tr>
<td>--datatype</td>
<td>Required</td>
<td>CellTower, Device, Geofence, Linkgeofence, LocationOptin, Orgunit, SubscriberOptin, SubscriberProfile, Subscriptions, Trace</td>
<td>No default value</td>
<td>No default value</td>
</tr>
<tr>
<td>--optintype</td>
<td>Optional</td>
<td>Blacklist, DR, External, Global, LANDS, LORS Opt-in type applies only if the data type is LocationOptin or SubscriberOptin.</td>
<td>No default value</td>
<td>No default value</td>
</tr>
<tr>
<td>--tracetype</td>
<td>Optional</td>
<td>Device, Location, Subscriber Trace type applies only if the data type is Trace.</td>
<td>No default value</td>
<td>No default value</td>
</tr>
<tr>
<td>--translate</td>
<td>Optional</td>
<td>0 or 1, false or 'false' true, no or yes Translate applies only if the data type is SubscriberOptin.</td>
<td>'false'</td>
<td>No default value</td>
</tr>
</tbody>
</table>

Example

```
update --file myfile.csv --datatype trace --tracetype true
```
This chapter presents the following topics:

Referencing RTISH commands ................................................................. 282
Basic RTISH operations ........................................................................... 282
rti create application ............................................................................... 283
rti declare exchange ............................................................................... 283
rti declare queue ..................................................................................... 284
rti delete exchange ............................................................................... 284
rti delete queue ..................................................................................... 284
rti deploy host ....................................................................................... 285
rti disable ingester protocol ................................................................. 285
rti disable instance ............................................................................... 285
rti disable ssh-key ................................................................................. 285
rti enable ingester protocol ................................................................. 286
rti enable instance ............................................................................... 286
rti enable ssh-key ................................................................................. 286
rti ext install ......................................................................................... 286
rti jmx invoke ......................................................................................... 287
rti license audit ...................................................................................... 287
rti license deploy ................................................................................... 288
rti list application types ....................................................................... 289
rti list env ............................................................................................. 289
rti list hosts ........................................................................................... 290
rti list instances .................................................................................... 290
Chapter 16: RTISH Commands

rti list mbeans.............................................................. 290
rti list processes......................................................... 290
rti list properties.......................................................... 291
rti list ssh-keys........................................................... 291
rti logs collect............................................................ 291
rti logs list................................................................. 293
rti logs show............................................................... 293
rti metrics deploy......................................................... 294
rti metrics describe...................................................... 294
rti metrics disable......................................................... 295
rti metrics enable .......................................................... 295
rti metrics list .............................................................. 296
rti metrics reload........................................................... 296
rti metrics undeploy......................................................... 297
rti orgunit list mode ....................................................... 297
rti orgunit <mode>.......................................................... 297
rti refdata connect........................................................ 298
rti refdata buildSpatialIndex ......................................... 299
rti refdata orgunit disable .............................................. 299
rti refdata orgunit read.................................................... 299
rti refdata orgunit upsert.................................................. 300
rti refdata outputformat upsert......................................... 300
rti refdata rebuildOptins ............................................... 301
rti refdata reversegeocode.............................................. 301
rti remove application.................................................... 301
rti rename host............................................................ 303
rti restart................................................................. 303
rti resync................................................................. 303
rti set env................................................................. 303
rti set property .................................................. 304
rti set ssh-key password ........................................ 304
rti show metrics .................................................. 304
rti show output ................................................... 304
rti start .......................................................... 305
rti status .......................................................... 305
rti stream deploy ............................................... 305
rti stream describe ............................................. 306
rti stream disable .............................................. 306
rti stream enable ............................................... 306
rti stream undeploy ........................................... 307
rti stream list .................................................... 307
rti stop .......................................................... 307
rti system connect ............................................. 308
rti system disconnect ......................................... 308
rti system servers ............................................. 308
rti system shutdown ......................................... 309
rti system start ............................................... 309
rti undeploy host .............................................. 310
rti unset env .................................................... 310
rti unset property ............................................. 310
RTISH commands

This section contains syntax information and examples for the RTISH commands, including some basic operational commands that do not require the `rti` prefix. The following sections describe the syntax and usage of the commands that you can run with the RTI shell utility (RTISH).

Basic RTISH operations

The following commands do not require the `rti` prefix.

Table 55. Basic RTISH operations

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
</table>
| `!`     | Execute any system shell command from the `rtish` prompt. Use a space after the `!` character. For example:  
  `! pwd`  
  `! ls` |
| `//`    | Set inline comment markers (start of line only). |
| `;`     | Set inline comment markers (start of line only). |
| `clear OR cls` | Clear the screen for the current terminal session. |
| `date`  | Display the local date and time. For example:  
  `rtish> date`  
  Tuesday, November 12, 2013 3:44:01 PM PST |
| `exit`  | Exit from the shell. |
| `help`  | List and describe all RTI commands, or show usage for a specific command. For example:  
  `rtish> help rti set property`  
  Keyword: rti set property  
  Description: Sets a property on a given RTI application instance  
  ... |
| `script` | Run a script from inside the RTI shell.  
  `script --file filename` |
| `sleep` | Set a wait time between commands, useful when you are executing a script. For example, when you start an RTI component, you may want to wait for a few seconds to be sure that the component is fully started. The following command, for example, waits for 5 seconds:  
  `sleep --delay 5000` |
| `system properties` | Show the current properties for the system shell. |
| `version` | Display the RTISH version. |
rti create application

Description
Create an application of the given type on all nodes.

Example
rti create application --type locato --name basic-locator-1
Application instance basic-locator-1 has been created

The application name rejects characters other than "]-].
You must specify the type and name; optionally, you can specify a description.

rti declare exchange

Description
Provision an AMQP exchange. The command includes the following parameters:

- address: Specify the IP address or host name of the RabbitMQ instance rather than retrieving the information from the distribution grid or provisioning instance.
- durable: Specify true if you are declaring a durable exchange.
- exchangeName (required): The exchange name.
- exchangeType (required): AMQP exchange type:
  - Direct
  - Fanout
  - Headers
  - Topic

Example
rti declare exchange --exchangeName Exchangel --exchangeType Direct

Complete syntax
rti declare exchange --exchangeName <Name> --exchangeType <Type> --durable <true/false> --address <IP address or host name>
**rti declare queue**

**Description**
Provision an AMQP queue. The command includes the following parameters:

- **address**: Specify the IP address or host name of the RabbitMQ instance rather than retrieving the information from the distribution grid or provisioning instance.
- **durable**: Specify `true` if you are declaring a durable exchange.
- **queueName (required)**: The queue name.

**Example**

```
rti declare queue --queueName Queue1
```

**Complete syntax**

```
rti declare queue --queueName <Name> --durable <true/false> --address <IP address or host name>
```

**rti delete exchange**

**Description**
Delete an AMQP exchange. The command includes the following parameters:

- **address**: Specify the IP address or host name of the RabbitMQ instance rather than retrieving the information from the distribution grid or provisioning instance.
- **exchangeName (required)**: The exchange name.

**Example**

```
rti delete exchange --exchangeName Exchange1
```

**Complete syntax**

```
rti delete exchange --exchangeName <Name> --address <IP address or host name>
```

**rti delete queue**

**Description**
Delete an AMQP queue. The command includes the following parameter:

- **queueName (required)**: The queue name.

**Example**

```
rti delete queue --queueName Queue1
```

**Complete syntax**

```
rti delete queue --queueName <Name>
```
rti deploy host

Description
Adds a server node to the RTI cluster. The destination directory is optional. The default location is `/opt/pivotal`. You can also specify the port number and a description. This port is the remote system's SSH port, which defaults to 22.

Example
rti deploy host --host 10.0.0.5 --dir /opt/pivotal --port 10335

rti disable ingester protocol

Description
Disable a protocol for one or more ingester instances. This command does not apply to other application instances.

Example
rtish> rti disable ingester protocol --instances ingester-one@* --protocol gb
Property gb.enabled set to false
Executed 'disable gb', successfully on: ingester-one@xxxyyy1.corp.abc.com status: 0

The protocol argument is optional. If you do not specify it, all protocols are disabled.

rti disable instance

Description
Disable an application instance.

Example
rtish> rti disable instance --instances basic-locator-1@xxxyyy1.corp.abc.com
Updated instances:
basic-locator-1 [locator] disabled

rti disable ssh-key

Description
Disable an SSH key.

Do not specify quotes around the path to the key.

Example
rti disable ssh-key --key /path/to/the/key
rti enable ingester protocol

**Description**
Enable a protocol for one or more ingester instances. This command does not apply to other application instances.

The `protocol` argument is optional. If you do not specify it, all protocols are enabled.

**Example**
```
rtish> rti enable ingester protocol --instances ingester-one@* --protocol gb Property gb.enabled set to true
Executed 'enable gb', successfully on: ingester-one@xxxyyy1.corp.abc.com status: 0
```

rti enable instance

**Description**
Enable instances of an application on specific hosts.

**Example**
```
rti enable instance --instances basic-locator-1@localhost
```

The `instances` argument accepts a comma-separated list of names. For example:
```
rti enable instance --instances ingestgrid-1@localhost, ingestgrid-2@localhost
```

This comma-separated syntax works for all the commands that accept instance names as arguments.

rti enable ssh-key

**Description**
Authenticate an SSH key.

**Example**
```
rti enable ssh-key --key /path/to/the/key
```

Do not specify quotes around the path to the key.

rti ext install

**Description**
Copy a gzipped tar ball file from the local file system to all deployed RTI hosts' `rti/ext` directory. It also automatically un-gzips and un-tars the file. Files installed with this command must have a `.tar.gz` extension. For example, if you have a Java `.jar` file
named `customCode.jar` with custom code that needs to be deployed to RTI, first create the gzipped tar ball as follows:

```
tar czf customCode.tar.gz /path/to/local/file/customCode.jar
```

Use file "customCode.tar.gz" as the "--file" argument to the `rti ext install` command.
The `<RTI Install Dir>/rti-t/ext/lib/customCode.jar` file is on every deployed host. The file will now be a permanent part of the RTI installation's image, and be automatically included on any subsequently deployments using RTISH "rti deploy host" command.

**Note:** The ".tar.gz" file extension is not required, but RTISH will execute the bash command "tar xzf <file>" on the provided file, and it must not throw an exception & must result in valid Java .jar file being produced.

**Command Syntax**

```
rti ext install --file <file_name_and_path.tar.gz>
```

**Example**

```
rtish> rti ext install --file /path/to/local/file/customCode.tar.gz
```

---

**rti jmx invoke**

**Description**

Invoke a method using Java Management Extensions (JMX).

To run this command, specify the RTI process ID, the MBean name, and the JMX method. Also, specify the values of the parameters for the method in the correct order. Typically, run this command only when RTI technical support or engineering request that you do so.

**Example**

For example, you can capture all threads (a stack trace) for a local RTI process.

```
rti jmx invoke --mbean java.lang:type=Threading --process 21203 --method dumpAllThreads --params true,true
```

---

**rti license audit**

**Description**

Audit all license files on the host on which it is run. It looks for files with the .lic extension in a fixed location only viz. `<rti-t>` folder where all RTI artifacts are installed. The command displays a report of all the licensed features available in the files found and for each feature it displays several important attributes as shown in the example below. Warnings about invalid files with expired features, incorrectly formatted features and altered files are displayed, as well as if no files are found. Manually remove files with expired features, incorrectly formatted features, as well as altered files on all hosts from your RTI deployment.
Syntax

rti license audit

Example

rtish> rti license audit
Auditing RTI license files....
Reporting on license files in: /opt/emc_rti_310/rti-t/.

==License Information==
<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Issued On</th>
<th>Start Date</th>
<th>Expiration/End Date</th>
<th>Grace Period</th>
<th>Core Count</th>
<th>SWID</th>
</tr>
</thead>
<tbody>
<tr>
<td>RTI_BASIC</td>
<td>DEMO_LICENSE</td>
<td>2-feb-2017</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>ACTIVATED TO License</td>
<td>Site Number: 12347777V</td>
</tr>
<tr>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>ACTIVATED TO License</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RTI_BASIC</td>
<td>PERMANENT</td>
<td>2-feb-2017</td>
<td>N/A</td>
<td>N/A</td>
<td>128</td>
<td>ELMRTI02176TBW</td>
<td>ACTIVATED TO License</td>
</tr>
<tr>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>ACTIVATED TO License</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RTI_BASIC</td>
<td>PERMANENT</td>
<td>2-feb-2017</td>
<td>N/A</td>
<td>N/A</td>
<td>96</td>
<td>ELMRTI0217X46C</td>
<td>ACTIVATED TO License</td>
</tr>
<tr>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>ACTIVATED TO License</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RTI_BASIC</td>
<td>SUBSCRIPTION</td>
<td>2-feb-2017</td>
<td>02-Feb-2017</td>
<td>02-Feb-2018</td>
<td>02-Feb-2018</td>
<td>64</td>
<td>ELMRTI0217BHW1</td>
</tr>
<tr>
<td>16-Feb-2018</td>
<td>N/A</td>
<td>N/A</td>
<td>ACTIVATED TO License</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RTI_BASIC</td>
<td>EVALUATION</td>
<td>2-feb-2017</td>
<td>N/A</td>
<td>2-may-2017</td>
<td>32</td>
<td>ELMRTI02172KPS</td>
<td>ACTIVATED TO License</td>
</tr>
</tbody>
</table>

***NOTE***:
Only valid license feature details are reported. Details are NOT reported for *expired* features, *incorrectly formatted* features, as well as for features in *altered* files!
Please REMOVE such files from the RTI installation.Audit complete.

rti license deploy

Description

Install the license file at a specific location on every host that is part of the RTI cluster. The file must be a valid license file; therefore, it must have a .lic extension. The installation location is fixed viz. the rti-t folder with the rest of RTI artifacts. The command does not install the file unless the rti-t folder is found. One file is deployed at a time. The command installs the license file on every host on which RTI is deployed including the host on which the command is run. This list of hosts can be found in the cluster-config.xml file under rti-t/common/rtish/etc/.
**Syntax**

```
rtish> rti license deploy --file
path_to_file/license_file_name.lic
```

The `--file` parameter takes a file name with an absolute or relative path. If the command cannot install the file it will display the error condition. In the example below, the command installs successfully on three out of four hosts and cannot install on the `rti-host4` because there is no `rti-t` folder.

**Example**

```
rtish> rti license deploy --file /home/test-user/2815219_02-Feb-2017.lic
Deployed license file on 3 out of 4 hosts.
Did not deploy on:
Host:rti-host4, Error:scp: /home/test-user/emc_rti_310/rti-
t/2815219_02-Feb-2017.lic: No such file or directory
```

**rti list application types**

**Description**

List available application types that you can create:

- `distgrid`: Distribution grid application
- `ingestgrid`: Ingest grid application
- `ingester`: Ingester application
- `locator`: Locator application
- `provisioning`: Provisioning Manager application
- `rest-api`: REST API application

Use these choices as keywords for `rti create application`.

**Example**

```
rti list application types
```

**rti list env**

**Description**

Return the values of environment variables for specified instances.

**Examples**

```
rti list env --instances basic-locator-1@
rti list env --instances basic-locator-1@ --name RTI_BASE
```
rti list hosts

**Description**
List the hosts (nodes) in the RTI cluster.

**Example**
```
rtish> rti list hosts
xxxxyyyy1.corp.abc.com (SSH Port: 22)
...
```

rti list instances

**Description**
Show all existing instances for all applications.

**Example**
```
rtish> rti list instances
--host localhost Host: localhost
basic-locator-1 [locator] enabled
...
```

rti list mbeans

**Description**
List MBeans in the specified JVM process.

```
rti list mbeans --process <process ID>
```

Use `--all` to include non-RTI MBeans in the list.

**Example**
```
rtish> rti list mbeans --process 12451
--all JMImplementation
com.sun.management
java.nio
java.lang
java.util.logging
```

rti list processes

**Description**
List all RTI processes on the local machine or all Java processes (`--all`) on the local machine.

**Example**
```
rtish> rti list processes
Local processes:
```
rti list properties

Description
Show the current configuration for one or more properties. You must specify an instance. The name argument is an optional property name.

Examples
rtish> rti list properties --instances ingest-grid-locator@* --name rti.locator.port
Properties of ingest-grid-locator@xxxyyy1.corp.abc.com:
rti.locator.port: 10334

To list all properties for an instance, do not specify a property name.

rtish> rti list properties --instances ingester-one@*
Properties of ingester-one@xxxyyy1.corp.abc.com:
rti.instance.name: ingester-one@xxxyyy1.corp.abc.com
iucs.tcp.port: 28003
rti.ingester.filters.blacklist: false
iucs.enabled: true
rti.ingester.filters.globalOptIn: false
gb.tcp.port: 28002
gb.enabled: true
...

rti list ssh-keys

Description
List SSH keys registered on the local machine.

Examples
rtish> rti list ssh-keys
SSH keys: PrivateKey [file = '/Users/user1/.ssh/id_rsa', protected = false,
             enabled = true]

rti logs collect

Description
Collect all the application and statistic logs from all deployed and enabled instances. The collected logs are copied to the local file system. The targetDir and name options are used to specify the location where the logs are saved. A compressed file that includes all the collected log files is generated with the file name specified by the name option.
Specify the following options to narrow down the logs to specific sub-sets:

- **targetDir**: (Mandatory) Local file system path to copy the logs and statistic logs.
- **Name**: (Mandatory) Directory name of the top level sub-directory under the target directory.
- **instanceType**: (Optional) Comma-delimited list of instance types to include. Valid values are ingester, ingestgrid, distgrid, locator, provisioning, rest-api, and hdfs-transport. If the instanceType is not specified, all types are included. Pressing Tab for this option will prompt with help for list of valid instanceType.
- **logType**: (Optional) Comma-delimited list of available log file types to include. Valid values are rti, gemfire, statistics, and all (rti, gemfire, statistics, stdout). If the logType is not specified, all types are included. Pressing Tab for this option will prompt with help for list of valid logType.
- **Instances**: (Optional) Comma-delimited list of RTI instance name(s) to list the logs/statistics from. Pressing Tab for this option will prompt with all the instance names.
- **hosts**: (Optional) Comma-delimited list of RTI host name(s) to list the logs/statistics from. This could be useful for certain types of issues, and provides a quick way to gather logs for all instances or a single (or list of) host(s). Pressing Tab for this option will prompt with all the host names.
- **dateRange**: (Optional) Date range in which the logs/statistics are collected. Format: "(yyyy-MM-dd HH:mm:ss, yyyy-MM-dd HH:mm:ss)". For example, "(2016-06-30 00:00:01, 2016-06-30 14:54:00)". Note that when specifying the date range, the double quotes are required. Use rtish logs list to check the date of log files.

**Examples**

The following example collects RTI logs from the distribution grid, and saves them to the target directory. In this example, target directory is a relative path to the current directory where the RTISH command is executed. The top-level directory is named as logs, which is under the target directory:

```
rtish> rtish log collect --instanceType distgrid --logType rti --targetDir target --name logs
Logs/statistics are collected to target directory target under top-level directory logs
```

This is the structure of the collected log files:

```
target
    ├── logs
    │    └── localhost
    │        └── distgrid-one
    │                │    └── rti-distgrid.log
    │                │    └── rti-distgrid.trace.log
    │                │    └── rti-stdout.log
```
rti logs list

Description
List all the application and statistic logs from all deployed and enabled instances. The following options can be specified to narrow down the logs to specific sub-sets:

- **InstanceType**: (Optional) Comma-delimited list of instance types to include. Valid values are ingester, ingestgrid, distgrid, locator, provisioning, rest-api, and hdfs-transport. If the instanceType is not specified, all types are included. Pressing Tab for this option will prompt with help for list of valid instanceType.

- **LogType**: (Optional) Comma-delimited list of available log file types to include. Valid values are rti, gemfire, statistics, and all (rti, gemfire, statistics, stdout). If the logType is not specified, all types are included. Pressing Tab for this option will prompt with help for list of valid logType.

- **Instances**: (Optional) Comma-delimited list of RTI instance name(s) to list the logs/statistics from. Pressing Tab for this option will prompt with all the instance names.

- **Hosts**: (Optional) Comma-delimited list of RTI host name(s) to list the logs/statistics from. This could be useful for certain types of issues, and provides a quick way to gather logs for all instances or a single (or list of) host(s). Pressing Tab for this option will prompt with all the host names.

Examples
The following example lists rti logs from the distribution grid:

```
rtish> rti log list --instanceType distgrid --logType rti
Log files for: distgrid-one@localhost:
-rw-r--r-- 1 liud17 CORP\Domain Users 1969834 14 Oct 16:23 rti-distgrid.log
-rw-r--r-- 1 liud17 CORP\Domain Users 0 13 Oct 18:52 rti-distgrid.trace.log
-rw-r--r-- 1 liud17 CORP\Domain Users 191 14 Oct 14:02 rti-stdout.log
```

rti logs show

Description
Show logs and statistics from all deployed hosts.

Use `rti logs list` to return the file names of log files for given instances.
Example

rtish> rti show log --instances provisioning-one@* --file
rti- provisioning.log --lines 10
Log files for: provisioning-one@xxxyyy1.corp.abc.com: 2014-01-16
15:24:15,073 [main] INFO
org.springframework.integration.monitor.IntegrationMBeanExporter
OU[] -
  Located managed bean
...
2014-01-16 15:24:15,392 [main] INFO
io.pivotal.rti.boot.spring.AbstractSpringInitializer OU[] -
  Application
  provisioning.app started OK.

rti metrics deploy

Description
Provide the ability to deploy a metric stream definition into the cluster. The command requires a JSON-formatted file of the metric definition (see Creating user-defined metrics streams). On system restarts, any deployed and enabled definitions are automatically restarted.

Command syntax

rti metrics deploy --definition <url_to_json_definition_file>

Where the URL JSON definition file can be one of file:/ or classpath:/ style URL

Example

rtish> rti metric deploy --definition
file:/path/to/definition/user-metric-stream.json
Deployed userStreamName metric stream. Starting up for event processing. userStreamName successfully started.

rti metrics describe

Description
Provide the metric stream attribute configuration.

Command syntax

rti metrics describe --name <nameOfMetricStream>

Example

rtish> rti metrics describe --name device
Attribute
Value
----------------------------------------
name
device
enabled
true
where cellTower == 175 ||
  mobileEquipment.manufacturer = 'APPLE'
groupBy
  mobileEquipment.tac
  having eps.attach.attempts >= 8000 && sms.mo.2g.fail.rate > 0.6
includePatterns
  [**]
excludePatterns
  []
exportFields
  [ref.manufacturer, ref.model, *]
refDataType
device
windowLength
  60000
eventTime
  EVENT_TIME
slidePeriod
  20000
groupBySizingHint
  50000
statisticsEnabled
  true

**rti metrics disable**

**Description**
Disable a metric stream real-time event processing flow. Any remaining events that have been consumed and are in a processing state are processed. It does not undeploy the stream.

**Command syntax**
rti metrics disable --definition <name>

**Example**
rtish> rti metrics disable --name device
device successfully disabled.

**rti metrics enable**

**Description**
Enable a metric stream to process real-time events for analytical processing. The definition must be deployed before this command can have any effect.
Chapter 16: RTISH Commands

Command syntax
rtish> rti metrics enable --name <name>

Example
rtish> rti metrics enable --name device
device successfully enabled.

rti metrics list

Description
List the deployed metric stream definitions. The command provides an optional filter parameter “state” to display all, enabled, or disabled definitions (see Creating user-defined metrics streams).

Command syntax
rti metrics list [--state <all | enabled | disabled>]

Example
rtish> rti metrics list

<table>
<thead>
<tr>
<th>Name</th>
<th>Active</th>
</tr>
</thead>
<tbody>
<tr>
<td>mobilenetwork</td>
<td>true</td>
</tr>
<tr>
<td>device</td>
<td>true</td>
</tr>
<tr>
<td>subscribergroup</td>
<td>true</td>
</tr>
<tr>
<td>celltower</td>
<td>true</td>
</tr>
</tbody>
</table>

rti metrics reload

Description
Reload a packaged default metric stream definition by name. The definition is deployed and enabled by default unless the ‘enabled’ flag has been set to false. A definition cannot be reloaded unless it has been undeployed from the system using the undeploy command.

Command syntax
rti metrics reload --name <name> --enabled <true | false>

Example
rtish> rti metrics reload --name device --enabled true
Metric stream device has been reloaded from default definition. device successfully started.
rti metrics undeploy

Description
Undeploy a metric stream definition from the system and frees all attached resources. The definition is completely removed from the system. If this is not the desired effect, use the disable command.

Command syntax
rti metrics undeploy --name <name>

Example
rtish> rti metrics undeploy --name device device successfully undeployed.

rti orgunit list mode

Description
Change the list mode for an (OU from whitelist to blacklist or the reverse. Available mode values include:

Whitelist: The RTI system includes an event for further processing if the IMSI for the event is currently listed in the whitelist for the OU. If no IMSI was ever added to the whitelist (or if the whitelist was removed using the rti refdata orgunit <mode> command), then no filtering is applied. Processing continues for all incoming events.

Blacklist: If no IMSI was ever added to the blacklist (or if the blacklist was removed using the rti refdata orgunit <mode> command), then no filtering is applied. Processing continues for all incoming events. If the IMSI for the event is currently listed in the blacklist, then the RTI system filters out the event from further processing.

Example
rti orgunit list mode --orgunit orgunit1 --mode whitelist

rti orgunit <mode>

Description
This command manages the whitelist/blacklist IMSI list for the OU.

rti orgunit mode action --orgunit orgunitName --file filepath

Specify the OU with the --orgunit argument. Specify the absolute path to a file containing newline-separated IMSIs with the --file argument. The --file argument is required for all actions.

The following table lists the available mode values:
Table 56.  RTISH orgunit command modes

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>whitelist</td>
<td>The RTI system includes an event for further processing if the IMSI for the event is currently listed in the whitelist for the OU. If no IMSI was ever added to the whitelist (or if the whitelist was removed. See the remove action, below), then no filtering is applied. Processing continues for all incoming events.</td>
</tr>
<tr>
<td>blacklist</td>
<td>If no IMSI was ever added to the blacklist (or if the blacklist was removed. See the remove action, below), then no filtering is applied. Processing continues for all incoming events. If the IMSI for the event is currently listed in the blacklist, then the RTI system filters out the event from further processing.</td>
</tr>
</tbody>
</table>

The following table lists the available action values:

Table 57.  RTISH orgunit command actions

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>replace</td>
<td>Replace the whitelist/blacklist with the contents of a file containing an IMSI list. If no IMSI list existed previously, then the IMSI list is created for the OU.</td>
</tr>
<tr>
<td>append</td>
<td>Add the file contents to an existing IMSI white/blacklist. If no IMSI list existed previously, then the IMSI list is created for the OU.</td>
</tr>
<tr>
<td>delete</td>
<td>Delete the file contents from any existing IMSI white/blacklist. If no IMSI list existed previously, then the IMSI list is created for the OU.</td>
</tr>
<tr>
<td>remove</td>
<td>Remove the whitelist or blacklist IMSI list entirely. When the active list is removed, incoming events are not filtered. Set the active list mode using the <code>rti refdata list mode</code> command.</td>
</tr>
<tr>
<td>list</td>
<td>List the current contents of the white or blacklist.</td>
</tr>
</tbody>
</table>

Example

```
rti orgunit whitelist append --orgunit orgunit1 --file filepath
```

**rti refdata connect**

**Description**

This command provides an alternative means of connecting to the distribution grid prior to loading organization units with the rti orgunit upsert command. Use the `rti refdata connect` command only in environments where the locator for the distribution grid has not been defined as an application with RTISH. For example, reference data files may be uploaded to an administration server, and the data administrator may have shell access to that server, but no privileges to access the deployed applications or production systems where RTI is running.

Example

```
rti refdata connect --host localhost --port 10335
```
The preferred method of connecting to the grid is to use the `rti system connect` command and specify the locator:

```
rti system connect --locator <name>
```

where `<name>` is an application instance for a distribution grid locator, such as `dist-grid-locator@xyz1.corp.abc.com`

### rti refdata buildSpatialIndex

**Description**

Force the cell tower reference data to be reindexed. By default, cell tower data that flows into the distribution grid is not indexed automatically. Use this command to index the data.

**Example**

```
rti refdata buildSpatialIndex --freeze true
```

### rti refdata orgunit disable

**Description**

Disable an OU and all of its subscriptions. Specify the identifier for the OU with the `--key` argument.

When you disable an OU, you can still read it with the `rti orgunit read` command; the OU is not removed from the system.

**Example**

```
rti refdata orgunit disable --key PIV_RND
Org unit [PIV_RND] disabled.
```

### rti refdata orgunit read

**Description**

List one or all of the OUs in the system, regardless of their state (enabled or disabled). Optionally, specify the identifier for an OU with the `--key` argument.

The list of OUs is returned in JSON format and enclosed in square brackets. Edit and reload the output with the `rti orgunit upsert` command. The `read` command is also useful as a way to return the API key for a given OU. See also Organization unit data.

**Example**

```
rti refdata orgunit read --key PIV_RND Org unit(s):
[
  {
    "id" : "PIV_RND",
    "organizationName" : "PIV", "organizationalUnitName" : "RND",
    "enabled" :
      false,
    "apiKey" : "05b57edaae7a8e943547c447ae38e016", "exchangeName" :
```

Chapter 16: RTISH Commands

EMC Real-Time Intelligence (RTI) 3.1.2 Administrators Guide

299
rti refdata orgunit upsert

Description
Provide an alternative approach to loading reference data for organization units. The upsert command inserts new organization units and updates existing records as appropriate. Existing reference data that does not match the keys in the data being loaded is left intact. The RDL update command and the rti refdata orgunit upsert command produce the same results; the difference in the two commands is that the rti refdata orgunit upsert command loads data in either CSV format or JSON format, while the RDL command loads data in only CSV format. See Organization unit data for details about both formats.

See also the rti refdata connect command.

Example
The following example reads reference data in JSON format from a file called orgunits.json:

rti refdata orgunit upsert --file ~/orgunits.json

rti refdata outputformat upsert

Description
Load a JSON format file that contains definitions of custom output formats. See
Configuring output streams.

Example
rti refdata outputformat upsert --file myoutputformat.txt

rti refdata rebuildOptins

Description
Cleans and rebuilds the external location and subscriber opt-in data in the distribution grid. When you set up a subscription, it has an expiration date. This command checks the internal indexes and removes any expired subscriptions. This cleaning optimizes runtime performance and should be run on a daily basis.

This command applies to the external opt-in data in the distribution grid only and must not be run against other components.

Syntax
rti refdata rebuildOptins --type <type> --simulate <true/false>

rti refdata reversegeocode

Description
Reverse geocode geofence keys to geospatial coordinates. The command includes an optional file parameter to specify a CSV containing the list of geofences to be reverse geocoded. To reverse geocode all geofences in the distribution grid, run the command without the optional file name. On successful completion, the command returns the count of geofences that were reverse geocoded.

Example
Reverse geocode all geofences in the system:
rti refdata reversegeocode

Complete syntax
rti refdata reversegeocode --file <csv filename>

Note: The rti shell must be connected to the distribution grid prior to running the rti reversegeocode command. For example: rti refdata connect --host localhost --port 10335

rti remove application

Description
Remove an application.
Chapter 16: RTISH Commands

Example

rtish> rti remove application --name provisioning-one
Executed 'stop', Instances not enabled for this operation:
provisioning-one@xxxyyy1.corp.abc.com
Application provisioning-one removed.
rti rename host

Description
Rename an RTI host. This command is useful when you configure a system while on a VPN connection that is subsequently dropped or no longer used. The recorded host name may no longer be accurate, causing a failure to connect, but you can use this command to solve the problem.

```bash
rti rename host --host <oldhostname> --name <newhostname>
```

Example
```
rtish> rti rename host --host usxxrumsbrml.corp.emc.com --name rtilaptop1 Renamed to rtilaptop1.
```

rti restart

Description
Restart all running RTI applications on a specific host or application instances on specific hosts. You can specify a host or an instance on a host.

Example
```
rtish> rti restart --instances ingester-one@* Executed 'restart', successfully on:
ingester-one@xxxyyy1.corp.abc.com status: 0
ingester.app is running with PID 36563; waiting up to 60s...
ingester.app is running with PID 37177
```

You can also specify multiple instances by using a comma-separated list.

rti resync

Description
Re-synchronizes the RTI configuration. It may be required if a configuration update (any property change) is not successfully propagated to all of the RTI hosts. For example, a network connection may be lost during an update.

Example
```
rti resync
```

rti set env

Description
Set the value of environment variables for specified instances.

Example
```
rti set env --instances basic-locator-1@192.168.0.100 --name RTI_OPTS --value "-Xmx2g -XX:PermSize=128m"
```
rti set property

**Description**
Set configuration properties for an instance. You can set the same properties for multiple instances in a single command.

This command supports various property name values; see Configuration Properties for lists of supported properties.

**Example**
rti set property --instances ingestgrid-1@localhost,ingestgrid-2@localhost
--name rti.gemfire.locators --value 127.0.0.1[10334]

rti set ssh-key password

**Description**
Sets the passphrase for a particular SSH key.

**Example**
rti set ssh-key password --key /path/to/the/key
Set the passphrase for /path/to/the/key:

rti show metrics

**Description**
Return JMX metrics (JMX MBean attribute names and values) for a specific Java process. Get the process ID first by using the rti list processes command. You can also specify a JSONPath (XPath for JSON) query. See Managing the RTI System.

**Example**
rti show metrics --process 12345 --all --query $

rti show output

**Description**
Show information about the current message output from the RTI system. Optionally, enter a routing key and a number of messages.

rti show output --key <routing key> --messages <number>

If the --messages argument is not specified, the command waits for new messages to be processed through the system. The command exits if you press Esc. The --messages option is useful when you are submitting the rti show output command in a .cmd script.
Example

The following is an example of the output for two messages:

Message cc52f8e6-a0b9-b789-3f8d-cf7938322c43:
  Routing key: DECODER.STREAM.ALL
  Headers:
  {decoder.messageHash=c5336ec81fe7e67ca533644dd01b189,
   decoder.eventCount=2, decoder.payloadFormat=null}
  Message size: 268
Message 5bd8c4cf-6809-ac65-028a-39f55cd09840:
  Routing key: DECODER.STREAM.ALL
  Headers:
  {decoder.messageHash=0929a93657cf5e63f2d266e70858ac21,
   decoder.eventCount=1, decoder.payloadFormat=null}
  Message size: 132

rti start

Description

Start all stopped RTI application instances or specific application instances.

rti start
rti start --host ...
rti start --instances ...

Examples

rti start --instances ingest-grid-locator@*,dist-grid-locator@
rti start --instances *@192.168.0.100

rti status

Description

Return the status of all RTI application instances, all instances on a host, or a specific instance.

rti status
rti status --host ...
rti status --instances ...

Example

rti status --instances ingest-grid-locator@*,dist-grid-locator@

rti stream deploy
Chapter 16: RTISH Commands

**Description**
Provide the ability to deploy a filtered stream definition in the cluster. The command requires a JSON-formatted file of the filter definition. See Filtering Output Streams for more details. On system restarts, any deployed and enabled filtered stream definitions are automatically restarted.

**Command syntax**
rti stream deploy --definition <url_to_json_definition_file>

Where the URL JSON definition file can be one of file:// or classpath:// style URL

**Example**
rtish> rti stream deploy --definition file:/path/to/definition/file.json

**rti stream describe**

**Description**
Output the definition of a previously deployed named filtered output stream. The definition must be deployed before this command can have any effect. See Filtering Output Streams for more details.

**Command syntax**
rtish> rti stream describe --name <name>

**Example**
rtish> rti stream describe --name example1

```json
{ "name" : "example1", "enabled" : true, "where" : "latitude > 0", "outputType" : "amqp", "amqpExchange" : "DECODER.STREAM", "amqpRoutingKey" : "DECODER.STREAM.FILTERED.EXAMPLE1" }
```

**rti stream disable**

**Description**
Disable a previously deployed and enabled named filtered output stream. The definition must be deployed and enabled before this command can have any effect. See Filtering Output Streams section for more details.

**Command syntax**
rtish> rti stream disable --name <name>

**Example**
rtish> rti stream disable --name example1

**rti stream enable**
**Description**

Enable a previously deployed named filtered output stream to filter real-time events. The definition must be deployed before this command can have any effect. See Filtering Output Streams section for more details.

**Command syntax**

```
rtish> rti stream enable --name <name>
```

**Example**

```
rtish> rti stream enable --name example1
```

---

**rti stream undeploy**

**Description**

Undeploy a filtered stream definition from the system and frees all associated resources. The definition is completely removed from the system. The command requires the name of a previously deployed named filter definition, see Filtering Output Streams section for more details.

**Command syntax**

```
rti stream deploy --name <name of previously deployed filtered stream>
```

**Example**

```
rtish> rti stream undeploy --name example1
```

---

**rti stream list**

**Description**

List all previously deployed named filtered output streams, along with their status. See Filtering Output Streams section for more details.

**Command syntax**

```
rtish> rti stream list
```

**Example**

```
rtish> rti stream list
example1    enabled
rtish>
```

---

**rti stop**

**Description**

Stop all running RTI application instances or specific application instances.
Important: The rti stop command is deprecated. Run the rti system shutdown command instead.

rti stop
rti stop --host ...
rti stop --instances ...

rti system connect

Description
Establish a persistent RTISH connection to the GemFire system (making RTISH a client). Use this command to select a locator before running provisioning commands in RTISH or using the rti orgunit upsert command.

rti system connect --locator <name>

where name is an application instance for a locator, such as:

dist-grid-locator@xyz1.corp.abc.com

Example
srtish> rti system connect --locator ingest-grid-locator@usxxnorwigm1.corp.emc.com

rti system disconnect

Description
Disconnect a persistent RTISH connection from the GemFire system.

Example
rtish> rti system disconnect --locator ingest-grid-locator@usxxnorwigm1.corp.emc.com

rti system servers

Description
List the connected servers.

Example
rti system servers
rti system shutdown

Description
Shut down all applications in the RTI system, in the appropriate order. Optionally, specify a delay in seconds and a number of retries. The delay defines a wait time between the shutdown of each application. The default delay is five seconds. The number of retries defines the number of attempts to shut down each application if the initial attempt fails. The default is three retries.

The shutdown order is as follows:

1. Ingester and provisioning applications
2. Ingest grid and distribution grid applications
3. Locator applications

Examples
rtish> rti system shutdown
rtish> rti system shutdown --retries 5
rtish> rti system shutdown --delay 10

You can also stop specific application instances. For example:

rtish> rti system shutdown --instances ingest-grid-locator@*,dist-grid-locator@*
rtish> rti system shutdown --instances *@192.168.0.100

rti system start

Description
Start all applications in the RTI system in the appropriate order. Optionally, specify a delay in seconds. This delay defines a wait time between the startup of each application. The default delay is five seconds.

The startup order is as follows:

1. Locator applications
2. Ingest grid and distribution grid applications
3. Ingester and provisioning applications

Note: The rti system shutdown command uses the reverse order.

The startup and shutdown priorities for an application are specified in the corresponding rtish.<appname>.properties file. These properties files are stored in the src directories for the applications. For example:

~/emc_rti_<version>/code/admin/rtish/rtish-locators/src/main/resources/ rtish-locator.properties
**Chapter 16: RTISH Commands**

**Examples**

```bash
rtish> rti system start
rtish> rti system start --delay 10
```

You can also start specific application instances. For example:

```bash
rtish> rti system start --instances ingest-grid-locator@*,dist-grid-locator@
rtish> rti system start --instances *@192.168.0.100
```

**rti undeploy host**

**Description**
Remove a host (node) from an RTI cluster.

**Examples**

```bash
rti undeploy host --host 10.0.0.5
```

If the host is inaccessible, you can still remove it by using the offline argument. For example:

```bash
rti undeploy host --host 192.168.0.100 --offline yes
```

**rti unset env**

**Description**
Remove an environment variable property for specified instances.

**Example**

```bash
rti unset env --instances .... --name ....
```

**rti unset property**

**Description**
Remove a property from a given RTI application instance.

**Examples**

```bash
rti unset property --instances .... --name ....
```
This chapter presents the following topics:

Referencing reference data formats ........................................................................312
Reference data formats

Reference data is loaded using the RDL or the Provisioning Manager. The RDL command-line interface accepts comma-separated text files, and the Provisioning Manager AMQP interface accepts AMQP messages. For administrative and diagnostic purposes, the RDL can load all of the data types, including those that may be more typically loaded in message format, such as subscriptions and geofences. Therefore, this section defines the reference data formats for all of the data types that RTI supports. Each data type is defined in terms of the fields for each record.

For information about the message formats for data typically loaded by the Provisioning Manager (geofences, linkgeofences, and subscriptions), see Provisioning Manager Message Interface for Loading Reference Data.

### Table 58. Reference data formats

<table>
<thead>
<tr>
<th>Reference data type</th>
<th>--datatype keyword in RDL command</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cell towers</td>
<td>CellTower</td>
</tr>
<tr>
<td>Mobile devices</td>
<td>Device</td>
</tr>
<tr>
<td>Geofences</td>
<td>Geofence</td>
</tr>
<tr>
<td>Link geofences (associations between geofences and</td>
<td>Linkgeofence</td>
</tr>
<tr>
<td>subscriptions)</td>
<td></td>
</tr>
<tr>
<td>LANDS opt-ins</td>
<td>LocationOptin (and an associated opt-in type)</td>
</tr>
<tr>
<td>Organization units</td>
<td>Orgunit</td>
</tr>
<tr>
<td>Subscriber opt-ins</td>
<td>SubscriberOptin (and an associated opt-in type)</td>
</tr>
<tr>
<td>Subscriber profiles</td>
<td>SubscriberProfile</td>
</tr>
<tr>
<td>Event subscriptions</td>
<td>Subscriptions</td>
</tr>
<tr>
<td>Trace filters</td>
<td>Trace (Subscriber, Location, and Device trace types)</td>
</tr>
</tbody>
</table>

**Cell tower data**

This data holds information about radio access network assets. Event stream pipelines consult this information to:

- Maintain a spatial index: reverse geocode network locations to geospatial coordinates
- Enrich subscriber network events with geospatial coordinates when available

Cell tower data is not high volume or high velocity and can be completely replaced (on a daily basis, for example).
**Cell tower record format**

The fields in each record are as follows. One record represents one cell tower.

<table>
<thead>
<tr>
<th>Field number</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Cell identifier</td>
<td>Numeric (CELL ID)</td>
</tr>
<tr>
<td>2</td>
<td>Cell name</td>
<td>Text literal</td>
</tr>
<tr>
<td>3</td>
<td>Centroid easting</td>
<td>Numeric (the easting of the cell centroid in OSGB36). NOT currently used. Leave empty.</td>
</tr>
<tr>
<td>4</td>
<td>Centroid northing</td>
<td>Numeric (the northing of the cell centroid in OSGB36). NOT currently used. Leave empty.</td>
</tr>
<tr>
<td>5</td>
<td>Site postcode</td>
<td>Text literal (the valid postal code of the cell tower)</td>
</tr>
<tr>
<td>6</td>
<td>Cell site latitude</td>
<td>Numeric (the WGS84 latitude of the cell tower site)</td>
</tr>
<tr>
<td>7</td>
<td>Cell site longitude</td>
<td>Numeric (the WGS84 longitude of the cell tower site)</td>
</tr>
<tr>
<td>8</td>
<td>Cell Site easting</td>
<td>Numeric (the easting of the cell site in OSGB36). NOT currently used. Leave empty.</td>
</tr>
<tr>
<td>9</td>
<td>Cell Site northing</td>
<td>Numeric (the northing of the cell site in OSGB36). NOT currently used. Leave empty.</td>
</tr>
<tr>
<td>10</td>
<td>Radii 90</td>
<td>Numeric (the 90% certainty radius in meters)</td>
</tr>
<tr>
<td>11</td>
<td>Location Area Code, Tracking Area Code</td>
<td>Numeric integer: The combination of the LAC) and CELL ID is used as the key for the cell. For a 4G cell tower, the field is interpreted as the Tracking Area Code (TAC).</td>
</tr>
<tr>
<td>12</td>
<td>Centroid Latitude</td>
<td>Numeric (the WGS84 latitude of the cell tower centroid) Optional: Defaults to site latitude if not available.</td>
</tr>
<tr>
<td>13</td>
<td>Centroid Longitude</td>
<td>Numeric (the WGS84 longitude of the cell tower centroid) Optional: Defaults to site longitude if not available.</td>
</tr>
<tr>
<td>14</td>
<td>Cell Type</td>
<td>Text Literal: 2G, 3G, 4G, 3GFEMTO, 4GFEMTO Optional. Defaults to NA if not available.</td>
</tr>
<tr>
<td>15</td>
<td>Radii 80</td>
<td>Numeric (the 80% certainty radius in meters) Optional: Defaults to radii 90 if not available.</td>
</tr>
<tr>
<td>16</td>
<td>Radii 70</td>
<td>Numeric (the 70% certainty radius in meters) Optional: Defaults to radii 90 if not available.</td>
</tr>
</tbody>
</table>
Example of a cell tower record

Here is an example of a valid cell tower record in CSV format, ready for loading. (This example wraps to a second line in this document but contains one complete record.)

45856,M35502730,,,YL0
5QP,90.0,180.0,,1000.0,244,90.0,180.0,NA,1000.0,1000.0

Device data

Device data holds information about available devices on the operator network. Event stream pipelines consult this information to:

- Maintain a device database, indexed using the TAC
- Enrich subscriber network events with device information, when the TAC is available

Device information is not high volume or high velocity and can be completely replaced (on a daily basis, for example).

Device record format

The fields in each record are as follows. One record represents one device.

<table>
<thead>
<tr>
<th>Field number</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Type Allocation Code (TAC)</td>
<td>Text literal of 8 digits (allotted by GSM-A) For example: 35896704</td>
</tr>
<tr>
<td>2</td>
<td>Manufacturer</td>
<td>Text literal from the GSM-A TAC Db For example: HTC</td>
</tr>
<tr>
<td>3</td>
<td>Model</td>
<td>Text literal. Optional: defaults to UNKNOWN For example: Desire S</td>
</tr>
<tr>
<td>4</td>
<td>deviceGeneration</td>
<td>Text literal. Optional: defaults to UNKNOWN. If specified by the user it must be one of &quot;2G&quot;, &quot;3G&quot;, &quot;4G&quot; (case-insensitive) For example: 2G or 4G</td>
</tr>
<tr>
<td>5</td>
<td>uaProfile</td>
<td>Text literal. Optional: defaults to UNKNOWN. Note the example in the sample Device Records below.</td>
</tr>
</tbody>
</table>

Examples of device records

Here are some examples of valid device records in CSV format, ready for loading.

01159800,Nokia,Nokia 2760,, 01159900,Huawei,Huawei T202,1
01160000,Motorola,Motorola W230a,1

In the first row, the complete "model" is Apple Ipad 2,0. The final comma in these examples is not a delimiter.
An event subscription corresponds to one event stream that is delivered to an OU.

Do not use the RDL to load subscriptions into an RTI system if the Provisioning Manager (PM) is being used by external users to create subscriptions. The data loaded by the RDL will overwrite any subscriptions that have been set up using the PM.

See also Event Subscription Messages.

### Event subscription record format

Table 61. Event subscription record format

<table>
<thead>
<tr>
<th>Field number</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Organization name</td>
<td>Text literal</td>
</tr>
<tr>
<td>2</td>
<td>Organization unit</td>
<td>Text literal</td>
</tr>
<tr>
<td>3</td>
<td>Subscription ID</td>
<td>Alphanumeric text: [a-z][A-Z][0-9][^-]. Must be unique for an organization/organization unit tuple.</td>
</tr>
<tr>
<td>4</td>
<td>Suspended</td>
<td>true or false</td>
</tr>
<tr>
<td>5</td>
<td>Effective from</td>
<td>Date and time format: yyyy-MM-dd’T’HH:mm:ss.SSSZ For example: 2013-12-01T23:59:59.999+0000</td>
</tr>
<tr>
<td>6</td>
<td>Effective to</td>
<td>Same as Effective from</td>
</tr>
<tr>
<td>7</td>
<td>tagIndex</td>
<td>A positive integer &gt;=10. This number is a unique system-wide index for the subscription; the index optimizes event processing. Use low numbers for this value, starting with 10. This value is required in the record and must be unique. The RDL user must ensure that it is unique.</td>
</tr>
</tbody>
</table>

### Examples of event subscription records

Here are some examples of valid subscription records, ready for loading. One row represents one event subscription.

VMW,EP,SUB01,true,2012-11-23T00:00:00.000+0000,2013-12-01T23:59:59.999+0000,17 VFUK,RND,SUB01,true,2012-11-23T00:00:00.000+0000,2013-12-01T23:59:59.999+0000,18
VFUK,RND,SUB02,true,2012-11-23T00:00:00.000+0000,2013-12-01T23:59:59.999+0000 VFUK,RND,SUB03,true,2012-11-23T00:00:00.000+0000,2013-12-01T23:59:59.999+0000
VMW,EP,SUB02,false,2012-11-23T00:00:00.000+0000,2013-01-07T20:15:59.999+0000,20 VMW,EP,SUB03,true,2012-11-23T00:00.000+0000,2013-12-07T20:15:59.999+0000

### Geofence and Linkgeofence (LGF) Data

Geofence data defines geographic areas of interest (polygons) that you can associate with event subscriptions in a linkgeofence (LGF). See also Geofence Messages and Linkgeofence Messages.
**Geofence record format**

The fields in each record are as follows. One record represents one geofence.

<table>
<thead>
<tr>
<th>Field number</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Organization name</td>
<td>Text literal</td>
</tr>
<tr>
<td>2</td>
<td>Organization unit</td>
<td>Text literal</td>
</tr>
<tr>
<td>3</td>
<td>Geofence ID</td>
<td>Alphanumeric text [a-z][A-Z][0-9][]. Must be unique for an organization/organizationunit tuple</td>
</tr>
<tr>
<td>4</td>
<td>SRID</td>
<td>Spatial reference identifier. A decimal number; the default is 4326 (WGS84 standard).</td>
</tr>
<tr>
<td>5</td>
<td>Geometry buffer</td>
<td>Default is 0 if left empty. Do not use this field unless you are instructed to do so by EMC.</td>
</tr>
<tr>
<td>6</td>
<td>Geometry (polygon)</td>
<td>Well Known Text (WKT) format-encoded geometry entity (Open GIS Simple Feature Access, Version 1.2.1).</td>
</tr>
<tr>
<td>7</td>
<td>Spatial Index Name</td>
<td>(Optional) Alphanumeric text [a-z][A-Z][0-9]. Case insensitive. Currently limited to the following values: BufferedRadii90 BR90 Default Index See Using Selectable Geocoding for more information.</td>
</tr>
</tbody>
</table>

**Examples of Geofence records**

Here are some examples of valid geofence records in CSV format, ready for loading.

```
VFUK,RND,NBY,,,POLYGON ((-1.58809 51.328949, -1.58809 51.563709, 0.98172 51.563709, 0.98172 51.328949, -1.58809 51.328949)),BufferedRadii90
VFUK,RND,MK,,,POLYGON ((-0.88737 51.969212, -0.88737 52.196331, -0.59182 52.196331, -0.59182 51.969212, -0.88737 51.969212)),BR90
VMW,EP,MK,,0,POLYGON ((-0.88737 51.969212, -0.88737 52.196331, -0.59182 52.196331, -0.59182 51.969212, -0.88737 51.969212))
```

**Linkgeofence record format**

The fields in each record are as follows. One record represents one linkgeofence.
Table 63.  Linkgeofence record format

<table>
<thead>
<tr>
<th>Field number</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Organization name</td>
<td>Text literal</td>
</tr>
<tr>
<td>2</td>
<td>Organization unit</td>
<td>Text literal</td>
</tr>
<tr>
<td>3</td>
<td>Subscription ID</td>
<td>Alphanumeric text [a-z][A-Z][0-9][.-].</td>
</tr>
<tr>
<td>4</td>
<td>Geofence ID</td>
<td>Alphanumeric text [a-z][A-Z][0-9][.-].</td>
</tr>
<tr>
<td>5</td>
<td>Replace</td>
<td>Must be unique for an organization/organization unit true or false</td>
</tr>
<tr>
<td>6</td>
<td>Simulate</td>
<td>true or false</td>
</tr>
</tbody>
</table>

Examples of Linkgeofence records

The following are examples of valid linkgeofence records in CSV format, ready for loading.

VMW,EP,SUB01,MK,false,false  
VFUK,RND,SUB01,NBY,true,false  
VFUK,RND,SUB03,MK,false,false

Location opt-in data

Location opt-in data is used as a filter to determine whether a network event qualifies for further processing or is filtered out, based on location information. This data holds network locations (LACs and TACs, cell IDs) and time intervals.

This data is low volume but high velocity. Following an initial bulk file submission, delta files can be resubmitted periodically.

This data can also be updated in real-time using the Provisioning Manager.

Location opt-in record format

The fields in each record are as follows. One record represents one location opt-in.

Table 64.  Location opt-in record format

<table>
<thead>
<tr>
<th>Field number</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Location area code (LAC)</td>
<td>Numeric (integral). The LAC where the subscriber can be tracked.</td>
</tr>
<tr>
<td>2</td>
<td>Cell identifiers</td>
<td>(Optional) Numeric or text literals. The cell identifiers where the subscriber can be tracked. Defaults to “*” (all cell IDs)</td>
</tr>
<tr>
<td>3</td>
<td>Tags</td>
<td>Delimited numeric integers. Optional; used for stream multiplexing; an opt-in can be forced to represent more than one stream subscription. Defaults to “*”. Contact your RTI representative if you want to use this advanced feature.</td>
</tr>
</tbody>
</table>
Chapter 17: Reference Data Formats

<table>
<thead>
<tr>
<th>Field number</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
</table>
| 4            | Time intervals  | (Optional) The time intervals during which the subscriber can be tracked. Defaults to "**". This is an embedded delimited field with a semicolon as the delimiter. The fields in order are: Time zone (Time Zone Database ID) Calendar flag (currently defaults to 6 and is not used) Interval tuple (start time|end time)  
For example: GB;6;17:00:00.001 | 18:00:00.000;18:30:00.00|19:00:00.000 |

The interval start and end times can also be expressed as a Java milliseconds timestamp: a decimal number, measured in milliseconds (not seconds) between the time and midnight, January 1, 1970 UTC.

**Examples of location opt-in records**
The following are examples of valid location opt-in records in CSV format, ready for loading.

51,*  
235,*  
54862,*  
236,*  
37470,*

**Organization unit data**
This data defines the entities that are entitled to request and receive event subscriptions. Each subscription passes in the name of a valid OU) as part of the credentials for the request. An OU consists of a user-defined organization name, a user-defined unit of work name, and the effective dates (when the OU starts and stops being active).

OU information is low volume and low velocity. Following an initial bulk load, delta updates can be loaded periodically.

You can load OU data in CSV format or JSON format. Additional options are available if you use the JSON format.

For details about how the OU data is used to submit event subscriptions, see Subscribing to events and Provisioning Manager Message Interface for Loading Reference Data.

**OU record format**
The fields in each record are as follows. One record represents one OU.

<table>
<thead>
<tr>
<th>Field number</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Organization name</td>
<td>Text literal: name of organization.</td>
</tr>
<tr>
<td>2</td>
<td>Organization unit</td>
<td>Text literal: name of organization unit.</td>
</tr>
<tr>
<td>Field number</td>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>----------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>3</td>
<td>Effective from</td>
<td>Text literal (ISO8601 format: yyyy-MM-dd'T'HH:mm:ss.SSSZ)</td>
</tr>
<tr>
<td>4</td>
<td>Effective to</td>
<td>Text literal (ISO8601 format: yyyy-MM-dd'T'HH:mm:ss.SSSZ)</td>
</tr>
<tr>
<td>5</td>
<td>Admin role</td>
<td>(Optional) true or false Whether the organization can submit reference data requests. If false, the organization can only create streaming requests. Defaults to false.</td>
</tr>
<tr>
<td>6</td>
<td>Reply to</td>
<td>(Optional) true or false Whether the organization can specify a replyTo header on its messages, providing control over which queue the Provisioning Manager replies to. Defaults to false.</td>
</tr>
<tr>
<td>7</td>
<td>API key</td>
<td>(Optional) Plain text API key can be generated either by an administrator outside of RTI, or automatically by RTI itself. This key must be verbally communicated to third parties who create event subscriptions. Any string may be used as the API key, and the administrator can generate it using a reasonably secure pseudo-random generator. Alternatively, the system-wide salt value can be used to generate the API key. If an external administrator assigns the keys to an OU, the following configuration property must be set to true: rti.distgrid.refdata.orgunit.allowExternalApiKeys is set to true. This property defaults to false. If the System generates an API key for an organization unit, to retrieve the value, use the REST API interface to obtain the generated value, and share the value with the third party. See Using the REST Interface to Inspect Data for details on use of the REST API interface.</td>
</tr>
<tr>
<td>8</td>
<td>Mode</td>
<td>whitelist (default): The RTI system includes the event for further processing if the IMSI for the event is currently listed in the whitelist for the OU. If no IMSI was ever added to the whitelist (or if the file contents of the whitelist was deleted), then no filtering is applied. Processing continues for all incoming events. blacklist: If no IMSI was ever added to the blacklist (or if the file contents of the blacklist was deleted), then no filtering is applied. Processing continues for all incoming events. If the IMSI for the event is currently listed in the blacklist, then the RTI system filters out the event from further processing.</td>
</tr>
</tbody>
</table>

**Examples of OU records in CSV format**

Here is an example of some valid OU records in CSV format, ready for loading.
The combination of the first two fields in OU records identify both the orgUnit value and the exchange value in event subscription messages, and are used as the prefix for the routingKey and responseQName:

**OU data in JSON format**

The JSON file must contain a valid JSON array of objects:

```json
ext.orgUnit = "BRN_DG"
ext.apiKey="6d051b96f732e6607b30f9b9d5caf39a"
ext.exchange="BRN.DG"
ext.routingKey="BRN.DG.request"
ext.responseQName="BRN.DG.response"
[ <JSON array of objects>
  { <First organization unit>
    ... } 
  { <Second organization unit>
    ... } 
]
```

Each OU is an object in the array, consisting of the following name/value pairs:

**Table 66. OU data in JSON format**

<table>
<thead>
<tr>
<th>Name/Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>Correlation ID for matching up errors. This value is not retained when the reference data is loaded.</td>
</tr>
<tr>
<td>organizationName</td>
<td>Text literal: name of organization.</td>
</tr>
<tr>
<td>organizationalUnitName</td>
<td>Text literal: name of the OU.</td>
</tr>
<tr>
<td>enabled</td>
<td>True or false: enables or disables subscriptions for the OU.</td>
</tr>
<tr>
<td>apiKey</td>
<td>The API key for the OU. By default, the API key cannot be set, and the attribute should be omitted or its value left empty. You can set this attribute only if the configuration property rti.distgrid.refdata.orgunit.allowExternalApiKey is set to true. This property defaults to false.</td>
</tr>
<tr>
<td>exchangeName</td>
<td>Exchange name: optional for new OUs; ignored for updates to existing OUs.</td>
</tr>
<tr>
<td>requestQueue: {queue_name}</td>
<td>Request queue name: optional for new OUs; ignored for updates to existing OUs. The queue name is nested and takes the form &lt;exchangeName&gt;.request.</td>
</tr>
</tbody>
</table>
### Name/Value

<table>
<thead>
<tr>
<th>Name/Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>responseQueue: (queue_name)</td>
<td>Response queue name: optional for new OUs; ignored for updates to existing OUs. The queue name is nested inside <code>&lt;exchangeName&gt;.response</code>.</td>
</tr>
<tr>
<td>controlQueue: (queue_name)</td>
<td>Control queue name: optional for new OUs; ignored for updates to existing OUs. The queue name is nested inside <code>&lt;exchangeName&gt;.control</code>.</td>
</tr>
<tr>
<td>controlQueue: (messageTtl)</td>
<td>Control queue message time-to-live: optional for new OUs; ignored for updates to existing OUs. This setting is a mechanism for setting the expiration of individual queues. For example: 7200000. This value is nested inside <code>&lt;exchangeName&gt;.control</code>.</td>
</tr>
<tr>
<td>deadLetterQueue: (queue_name)</td>
<td>Dead letter queue name: optional for new OUs; ignored for updates to existing OUs. The queue name is nested inside <code>&lt;exchangeName&gt;.deadletter</code>. The default naming scheme is <code>&lt;exchangeName&gt;.deadletter</code>.</td>
</tr>
<tr>
<td>allowReplyTo</td>
<td><code>true</code> or <code>false</code>. Whether the organization can specify a <code>replyTo</code> header on its messages, providing control over which queue the Provisioning Manager replies to. Optional: defaults to <code>false</code>.</td>
</tr>
<tr>
<td>controlMessagesEnabled</td>
<td><code>true</code> or <code>false</code>. Whether control messages are completely enabled or disabled. Optional: defaults to <code>true</code>. Control messages communicate information about active subscriptions. For example, control messages are used to map a raw IMSI or an actual phone number to the hashed IMSI that is sent on the subscription stream; without these messages, the creator of the subscription will not be able to identify the phone numbers that belong to the hashed values. To disable control messages, you have to use a JSON format load with <code>controlMessagesEnabled</code> set to <code>false</code>.</td>
</tr>
<tr>
<td>isAdminRole</td>
<td><code>true</code> or <code>false</code>. Whether the organization can submit reference data requests. If false, the organization can only create streaming requests. Optional: defaults to <code>false</code>.</td>
</tr>
<tr>
<td>effectiveFromDate</td>
<td>Text literal (ISO8601 format: <code>yyyy-MM-dd'T'HH:mm:ss.SSSZ</code>). For example: <code>1970-01-01T01:00:00.000+0100</code> If this date is a future date, the OU is not yet active. It will become active past this date, if <code>enabled</code> is set to <code>true</code>.</td>
</tr>
<tr>
<td>effectiveToDate</td>
<td>Text literal (ISO8601 format: <code>yyyy-MM-dd'T'HH:mm:ss.SSSZ</code>). For example: <code>2016-01-01T23:59:59.999+0000</code> Setting this value to a date in the past disables an OU (as does the <code>enabled:false</code> attribute). You cannot delete OUs.</td>
</tr>
</tbody>
</table>
**JSON file examples**

The following example is a minimal JSON file that represents a single organization unit.

```json
[
  {
    "id" : "PIV_RND",
    "organizationName" : "PIV", "organizationalUnitName" : "RND",
    "enabled" : true,
    "effectiveFromDate" : "1970-01-01T01:00:00.000+0100",
    "effectiveToDate" : "2016-12-01T23:59:59.999+0000"
  }
]
```

The second example also represents a single organization unit but contains more attributes.

```json
[
  {
    "id" : "PIV_MKT",
    "organizationName" : "PIV", "organizationalUnitName" : "MKT",
    "enabled" : true,
    "exchangeName" : "PIV.MKT", "requestQueue" : {
      "queueName" : "PIV.MKT.request"
    },
    "responseQueue" : {
      "queueName" : "PIV.MKT.response"
    },
    "controlQueue" : {
      "queueName" : "PIV.MKT.control", "messageTtl" : 7200000
    },
    "deadLetterQueue" : {
      "queueName" : "PIV.MKT.deadletter"
    },
    "allowReplyTo" : true, "controlMessagesEnabled" : true,
    "isAdminRole" : false,
    "effectiveFromDate" : "1970-01-01T01:00:00.000+0100",
    "effectiveToDate" : "2016-12-01T23:59:59.999+0000"
  }
]
```
Subscriber opt-in data

Subscriber opt-in data is used as a filter to determine whether a subscriber event qualifies for further processing or is filtered out, based on IMSI values. This data holds the subscriber ID (anonymized IMSI), network locations, and time intervals.

This data is high volume and high velocity. Following an initial bulk file submission, delta files can be submitted periodically. You must reload this data in full when the secure SALT is changed on the platform.

This data can also be updated in real-time using the Provisioning Manager.

**Subscriber opt-in record format**

The fields in each record are as follows. One record represents one subscriber opt-in.

**Table 67. Subscriber opt-in record format**

<table>
<thead>
<tr>
<th>Field number</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Subscriber identifier</td>
<td>Numeric. The raw IMSI key.</td>
</tr>
<tr>
<td>2</td>
<td>Location area codes (LACs)</td>
<td>Delimited integers or text literals. The LACs where the subscriber can be tracked. Optional: defaults to &quot;*&quot; (all LACs)</td>
</tr>
<tr>
<td>3</td>
<td>Cell identifiers</td>
<td>Numeric or text literals. The cell identifiers where the subscriber can be tracked. Optional: defaults to &quot;*&quot; (all LACs)</td>
</tr>
<tr>
<td>4</td>
<td>Tags</td>
<td>Delimited numeric integers. Optional; used for stream multiplexing; an opt-in can be forced to represent more than one stream subscription. Defaults to &quot;*&quot;. Contact your RTI representative if you want to use this advanced feature.</td>
</tr>
<tr>
<td>5</td>
<td>Time intervals</td>
<td>The time intervals during which the subscriber can be tracked. Optional. Defaults to &quot;*&quot; This is an embedded delimited field with a semicolon as the delimiter. The fields in order are: Time zone (Time Zone Database ID) Calendar flag (currently defaults to 6 and is not used) Interval tuple (start time</td>
</tr>
</tbody>
</table>

The interval start and end times can also be expressed as a Java milliseconds timestamp: a decimal number, measured in milliseconds between the time and midnight, January 1, 1970 UTC.

**Examples of subscriber opt-in records**

Here are some examples of valid subscriber opt-in records in CSV format, ready for loading.

b61fe989172d6bbff33446544eaf1d28,*,*
2c77ed56f11d22bbf547d72065af807,*,*
90cf40e0868f42d6d439e50f1016bf95,*,*
Subscriber profile data contains subscriber IMSI and MSISDN values. RTI uses the IMSI value, but downstream consumers may make use of the MSISDN to submit subscriber information for the linked IMSI. Event stream pipelines consult this information to:

- Enrich subscriber network events, when the MSISDN is not available in the source event
- Perform title translation during provisioning and when a subscriber opt-in is submitted using the MSISDN

Device information is not high velocity but is high volume and should be completely replaced (on a daily basis, for example).

RTI is capable of maintaining a mapping between IMSI values and their paired TMSI values. See the ingester and ingest grid sections in Configuration Properties.

**Subscriber profile record format**

The fields in each record are as follows. One record represents one MSISDN.

**Table 68. Subscriber profile record format**

<table>
<thead>
<tr>
<th>Field number</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>IMSI</td>
<td>Text literal. Raw IMSI conforming to ITU E.212. For example: 404685505601234</td>
</tr>
<tr>
<td>2</td>
<td>MSISDN</td>
<td>Text literal in the E.165 format (long format). For example: 447541004706</td>
</tr>
<tr>
<td>3</td>
<td>Home post code</td>
<td>Text literal (optional). For example: RG14 2FN</td>
</tr>
<tr>
<td>4</td>
<td>Subscriber Groups</td>
<td>Delimited alphanumeric [a-z][A-Z][0-9][^-] text literal (optional). One or more subscriber group names that the subscriber belongs to. Group names are delimited by a semicolon (;). Case insensitive. Duplicate names ignored. The number of groups per subscriber cannot be greater than max.number.subscriber.groups (the default value is 128 characters). The length of the group name cannot be greater than max.length.subscriber.group.name (the default is 8 characters).</td>
</tr>
<tr>
<td>5</td>
<td>External Identifier</td>
<td>Text literal (optional). Identifies the subscriber to an external system.</td>
</tr>
</tbody>
</table>
**Examples of subscriber profile records**

Here are some examples of valid subscriber profile records in CSV format, anonymized.

```
lddef66028534a0, 60931f88-29df-4fe7-b2bb-1556afce3ed7
36721b77414f4c9, ba2a0563-837b-4d11-8f25-77ef53f86530
c79789fd3e8b474,
   dac4d4ee-8159-4c47-96e8-
15a4070959f6f,10011,Group1;Group2;Group3;externalId-1
e3fdec0aece71419, bb297d5d-e502-4e0-e-ae9-0e004d1c0378,,Group2,Group3
faab61850fb5474, 7b18c9e9-b848-4ed-bf89-
838787a4a814,,externalId-2 4b5bb42d85882, 8c7eacf1-c3c8-4978-a0aa-f3a7e99cb332
```

**Trace data**

Trace data turns on tracing for devices, locations, or subscribers. All events originating from a given device, location (LAC), or subscriber may be tagged for tracing.

**Device trace data**

Device trace data defines subscriber devices, identified by IMEI keys that are tagged for tracing. This data is low volume and low velocity. Following an initial bulk file submission, you can submit delta files periodically. You must reload all of this data when the secure SALT is changed on the platform.

**Device trace record format**

The fields in each record are as follows. One record represents one device trace.

<table>
<thead>
<tr>
<th>Field number</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Device identifier</td>
<td>Numeric (integral). The raw IMEI key.</td>
</tr>
<tr>
<td>2</td>
<td>Intercept flag</td>
<td>Flag to determine the interception mode (lawful intercept). The following case-sensitive values are supported: true/false (default) The value defaults to false unless you specify a value of true.</td>
</tr>
<tr>
<td>3</td>
<td>Location area codes (LACs)</td>
<td>(Optional) Delimited integers, or *** (all LACs, the default). The LACs where the device can be tracked.</td>
</tr>
<tr>
<td>4</td>
<td>Cell identifiers</td>
<td>(Optional) Numeric, or *** (all cell IDs, the default). The cell identifiers where the device can be tracked.</td>
</tr>
</tbody>
</table>
Chapter 17: Reference Data Formats

**Examples of device trace records**

Here are some examples of valid device trace records in CSV format, ready for loading.

357663017768556,false,*,*
352936001349772,true,*,*
357805023984442,true
354692042853693,true
357613004448485,false,64,6,*
49015420323751,false,987,78441

**Location trace data**

Location trace data defines subscriber locations, identified by LAC/TAC and cell IDs that are tagged for tracing. This data is low volume and low velocity. Following an initial bulk file submission, you can submit delta files periodically. You must reload this data in full when the secure SALT is changed on the platform.

**Location trace record format**

The fields in each record are as follows.

<table>
<thead>
<tr>
<th>Field number</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Location identifier</td>
<td>Numeric (integral). The raw LAC/TAC.</td>
</tr>
<tr>
<td>2</td>
<td>Intercept flag</td>
<td>Flag to determine the interception mode (lawful intercept). The following case-sensitive values are supported: true false (default) The value defaults to false unless you specify a value of true.</td>
</tr>
<tr>
<td>4</td>
<td>Cell identifiers</td>
<td>Numeric, or &quot;*&quot; (all cell IDs, the default). The cell identifiers where the subscriber can be tracked. Optional</td>
</tr>
</tbody>
</table>

**Examples of trace location records**

698,false,
79831 766,
true,84502
99,true 792,false,*

**Subscriber trace data**

Subscriber trace data defines subscribers, identified by IMSI, that are tagged for tracing. This data is low volume and low velocity. Following an initial bulk file submission, you can submit delta files periodically. You must reload all of this data when the secure SALT is changed on the platform.
**Subscriber trace record format**

The fields in each record are as follows.

Table 71. Subscriber trace record format

<table>
<thead>
<tr>
<th>Field number</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Subscriber identifier</td>
<td>Numeric (integral). The raw IMSI. You can override the global setting to hash the subscriber identifier using the <code>subscriber.trace.anonymize</code> property.</td>
</tr>
<tr>
<td>2</td>
<td>Intercept flag</td>
<td>Flag to determine the interception mode (lawful intercept). The following case-sensitive values are supported: true, false (default) The value defaults to false unless you specify a value of true.</td>
</tr>
<tr>
<td>3</td>
<td>Transaction pattern</td>
<td>(Optional) Text literal (a Java regular expression) Defaults to &quot;.:*&quot; (all transactions).</td>
</tr>
<tr>
<td>4</td>
<td>Location area codes (LACs)</td>
<td>(Optional) Delimited integers, or &quot;.:*&quot; (all LACs, the default) The LACs where the subscriber can be tracked.</td>
</tr>
<tr>
<td>5</td>
<td>Cell identifiers</td>
<td>(Optional) Numeric, or &quot;.:*&quot; (all cell IDs, the default) The cell identifiers where the subscriber can be tracked.</td>
</tr>
</tbody>
</table>

**Examples of trace subscriber records**

<table>
<thead>
<tr>
<th>Subid</th>
<th>Intercept</th>
<th>Transaction</th>
<th>Location area codes</th>
<th>Cell identifiers</th>
</tr>
</thead>
<tbody>
<tr>
<td>234151234567890</td>
<td>false</td>
<td>.:*</td>
<td>*</td>
<td></td>
</tr>
<tr>
<td>234150987654321</td>
<td>true</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>234155678901234</td>
<td>false</td>
<td>.:*</td>
<td>984, 78931</td>
<td></td>
</tr>
<tr>
<td>404685505601234</td>
<td>true</td>
<td></td>
<td></td>
<td>984, 78931</td>
</tr>
<tr>
<td>234151234098765</td>
<td>false</td>
<td>.:*</td>
<td>99</td>
<td></td>
</tr>
</tbody>
</table>
Chapter 18: Loading Reference Data with RTISH

This chapter presents the following topics:

- Loading reference data with RTISH ................................................................. 329
- upsert command ................................................................................................. 329
- delete command .................................................................................................. 330
- Command line arguments .................................................................................... 330
- Operational commands ....................................................................................... 331
- Tuning options ..................................................................................................... 332
Loading reference data with RTISH

Reference data can be loaded into the RTI system using the RTISH utility. RTISH also supports certain operations on loaded reference data as described below. We recommend using the RTISH utility in place of the RDL utility.

Below is a list of all reference data types supported by RTISH and the type of grid into which it can be loaded. For details about the reference data type (fields, formats), refer to the detailed command documentation that follows.

Table 72. Reference data types supported by RTISH

<table>
<thead>
<tr>
<th>Reference data type name</th>
<th>Ingest grid</th>
<th>Distribution grid</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cell Tower</td>
<td>Supported</td>
<td>Supported</td>
</tr>
<tr>
<td>Mobile Equipment</td>
<td>Supported</td>
<td>Supported</td>
</tr>
<tr>
<td>Subscriber Profile</td>
<td>Supported</td>
<td>Supported</td>
</tr>
<tr>
<td>Organizational Unit</td>
<td>Not Supported</td>
<td>Supported</td>
</tr>
<tr>
<td>Event Stream Subscriptions</td>
<td>Not Supported</td>
<td>Supported</td>
</tr>
<tr>
<td>GeoFence</td>
<td>Not Supported</td>
<td>Supported</td>
</tr>
<tr>
<td>Location Optin</td>
<td>Supported</td>
<td>Supported</td>
</tr>
<tr>
<td>Subscriber Optin</td>
<td>Supported</td>
<td>Supported</td>
</tr>
</tbody>
</table>

Note: The Trace datatype (and its subtypes, Device, Location and Subscriber) is no longer supported via RTISH.

Connecting to the system before executing RTISH refdata commands

Important: After bringing up RTISH the user must connect to a grid using the `rti refdata connect` command. Details can be found in the RTISH command reference: `rti refdata connect`.

upsert command

Description

Load data from the specified CSV file for the specified reference data type. It can be used to load new records (create) or update existing records by including the updated CSV record in the file.

Syntax

```bash
rti refdata <datatype> upsert --file <path to csv file> --translate <true or false> --optintype <optintype>
```

Examples

```bash
rti refdata celltower upsert --file path_name/cell_tower.csv
rti refdata locationoptin upsert --optintype lands --file
```
location_optin.csv rti refdata eventstreamsubscriptions upsert --file path_name/ess.csv

delete command

Description
Delete data records provided in the specified CSV file for the specified reference data type on the grid. If no file is specified then all records for the specified reference data type will be deleted on the grid. RTISH will confirm if all records should be deleted. To force the deletion of all records set --force to "true".

Syntax
rti refdata <datatype> delete --file <path to csv file> --translate <true or false> --optintype <optintype> --force <true or false>

Examples
rti refdata celltower delete --file path_name/cell_tower.csv
rti refdata locationoptin delete --optintype lors --file path_name/location_optin.csv
rti refdata subscriberoptin delete --optintype global --force true

Command line arguments

Description
RTISH commands support the following options. To view these options, the Tab key. Pressing Tab after the option will also display the available values.

Table 73. RTISH command options

<table>
<thead>
<tr>
<th>Argument</th>
<th>Required/optional</th>
<th>Values</th>
<th>Default (if not specified)</th>
</tr>
</thead>
<tbody>
<tr>
<td>datatype</td>
<td>Required</td>
<td>celltower mobileequipment subscriberprofile orgunit eventstreamsubscriptions geofence locationoptin subscriberoptin</td>
<td>None None</td>
</tr>
<tr>
<td>--file</td>
<td>Optional</td>
<td>Path to CSV file</td>
<td>None None</td>
</tr>
</tbody>
</table>
### Argument Table

<table>
<thead>
<tr>
<th>Argument</th>
<th>Required/optional</th>
<th>Values</th>
<th>Default (if not specified)</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>--optintype</code></td>
<td>Optional</td>
<td>global, blacklist, external, lands, lors</td>
<td>None</td>
</tr>
<tr>
<td><code>--translate</code> (only applies to Subscriber Optin)</td>
<td>Optional</td>
<td>true, false, 0, 1, yes, no</td>
<td>false</td>
</tr>
<tr>
<td><code>--force</code> (Note: eventstreamsubscriptions and geofence data types do not support the force delete option)</td>
<td>Optional</td>
<td>true or false</td>
<td>false</td>
</tr>
</tbody>
</table>

**Note:** "dr" opt-in type is no longer supported by RTISH though it is provided as a tab complete value.

**Note:** Refer to the RTISH commands section for a few commands specific to the Organizational Unit (OrgUnit) datatype, namely read and disable. Links are in the Operational commands section below.

### Operational commands

#### Description

These commands execute certain operations on the loaded data in the grid.

#### Linking event stream subscriptions to Geofences

The same command to upsert the eventstreamsubscriptions datatype also links the ESS provided in the CSV file with one or more existing geofences.

#### Example

```
rti refdata evenstreamsubscriptions upsert --file path_to_file/ess.csv
```
Chapter 18: Loading Reference Data with RTISH

The CSV record looks like this:

```
#orgName,ouName,subscriptionid,active,effectiveFrom,effectiveTo,tagIndex,
#     outputFormat,geofenceids
VMW,EP,SUB10,true,1353628800000,1385942399999,15,TEST,MK;NBY
```

In the above example the ESS “SUB10” is linked to the existing Geofences “MK” and “NBY”.

Also, the “delete” command unlinks the ESS from any linked Geofences.

**Additional operational commands**

Several other operational commands such as reverse geo coding, building spatial indexes, rebuilding optins, and so on are documented in the RTISH commands section:

- "rti refdata connect" — rti refdata connect
- "rti refdata buildSpatialIndex" — rti refdata buildSpatialIndex
- "rti refdata orgunit disable" — rti refdata orgunit disable
- "rti refdata orgunit read" — rti refdata orgunit read
- "rti refdata rebuildOptins" — rti refdata rebuildOptins
- "rti refdata reverseGeocode" — rti refdata reversegeocode

**Tuning options**

**Description**

These tuning options can optimize reference data loading. They are especially important for larger data sets.

Note that because RTISH only has commands to configure properties for RTI application instances (and RTISH is not itself considered to be one), these properties need to be set. Set them either by editing the local RTISH config.properties file directly, or by assignment using a System Property in the RTI_OPTS environment variable, following the pattern where "-D" is prepended to the property name.
### Table 74. RTISH tuning options

<table>
<thead>
<tr>
<th>Tuning option</th>
<th>Default value</th>
<th>Configuration method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rtish.refdata.batch.size</td>
<td>2500</td>
<td>Edit common/rtish/etc/config.properties on the host from which you execute RTISH to load data. Alternatively, set as system property using RTI_OPTS.</td>
<td>The batch size used by each processing thread.</td>
</tr>
<tr>
<td>rtish.refdata.batch.threads</td>
<td>16</td>
<td>Edit common/rtish/etc/config.properties on the host from which you execute RTISH to load data. Alternatively, set as system property using RTI_OPTS.</td>
<td>Controls the number of parallel loading threads.</td>
</tr>
</tbody>
</table>

**Example for configuring tuning options using RTI_OPTS**

Before launching the RTISH command line tool (in either interactive or batch command processing modes), execute the following bash command:

```bash
export RTI_OPTS="-Drtish.refdata.batch.size=2500 -Drtish.refdata.batch.threads=32"
```

**Note:** The parallelization and batching control for the early access version of RTISH RDL has only been enabled for Subscriber Opt-in, Geofence and Subscriber Profile data types. These are typically the largest reference data record sets, so the limitation is not expected to be an issue.
This chapter presents the following topics:

Overview ........................................................................................................................................335
Common message elements ........................................................................................................336
Overview

Reference data is loaded using the RDL or the Provisioning Manager. The RDL command-line tool accepts comma-separated text files, and the Provisioning Manager interface accepts AMQP messages. This section defines the reference data formats for the data types that the Provisioning Manager supports, defined in terms of the data elements for each message. For information about the formats for data loaded by the RDL, see the Reference Data Formats section.

All messages (submission and response) have some common elements, which are described first in this section.

Table 75. Provisioning manager messages

<table>
<thead>
<tr>
<th>Message type</th>
<th>Submission message name</th>
<th>Response message name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Geofence setup</td>
<td>PROV.01.GF.SUBMIT</td>
<td>PROV.01.GF.RESPONSE</td>
</tr>
<tr>
<td>Geofence stream association (linkgeofence)</td>
<td>PROV.01.LGF.SUBMIT</td>
<td>PROV.01.LGF.RESPONSE</td>
</tr>
<tr>
<td>Geofence stream activation</td>
<td>PROV.01.SST.SUBMIT</td>
<td>PROV.01.SST.RESPONSE</td>
</tr>
<tr>
<td>Event (stream) subscription</td>
<td>PROV.01.ESS.SUBMIT</td>
<td>PROV.01.ESS.RESPONSE</td>
</tr>
<tr>
<td>Stream subscription mapping</td>
<td>PROV.01.SSM.SUBMIT</td>
<td>PROV.01.SSM.RESPONSE</td>
</tr>
<tr>
<td>Stream subscription opt-in</td>
<td>PROV.01.SSO.SUBMIT</td>
<td>PROV.01.SSO.RESPONSE</td>
</tr>
<tr>
<td>Stream subscription opt-in mapping</td>
<td>N/A</td>
<td>PROV.01.SSO.CONTROL</td>
</tr>
</tbody>
</table>

In addition to the streams of event messages themselves, the Provisioning Manager interface uses three basic message flows:

- Provisioning request messages, as initiated by the event subscriber to set up data model entities.
- Provisioning response messages, as sent by the RTI system in response to request messages.
- Provisioning control messages, as initiated by RTI to communicate information about active subscriptions.

All messages submitted to RTI generate a response. All provisioning request submissions are submitted to the default virtual host “/”.

- AMQP exchange name: orgName.ouName
- Request message submission routing key: orgName.ouName.request (sent to a virtual host’s exchange with a routing key)
- Response message processing queue: orgName.ouName.response
- Control message processing queue: orgName.ouName.control
- Dead letter message queue: orgName.ouName.deadletter
Chapter 19: Provisioning Manager Message Interface for Loading Reference Data

Here \textit{orgName} is the organization identifier and \textit{ouName} is the organization unit identifier already loaded into RTI (as OU reference data).

Periodically the platform may submit control messages that provide control or operational information about the ESS. These messages can be consumed from the organization’s control queue. These messages are specific, tailored to an operating entity’s environment, and documented separately. By default, no control messages will be delivered. Examples of control messages can be:

- Periodic changes to customer identifier to anonymization mapping
- Operational metrics on the event stream.

The dead letter queue contains a copy of submission messages that are deemed to be in error and have been rejected by RTI. This queue is for operational and support tools to track rejected messages.

**Common message elements**

The table below lists the mandatory data elements and the data type required for all of the messages. Absence of these data elements will result in a validation failure with the message being rejected by the RTI system.

**Table 76. Common message elements**

<table>
<thead>
<tr>
<th>Number</th>
<th>Name</th>
<th>Message location</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>userId</td>
<td>Content class header</td>
<td>The Rabbit MQ user ID. Fixed as \textit{orgIdentifier_ouIdentifier}</td>
</tr>
<tr>
<td>2</td>
<td>content_encoding</td>
<td>Content class header</td>
<td>The content encoding used to encode the message payload. Default: UTF-8</td>
</tr>
<tr>
<td>3</td>
<td>message_id</td>
<td>Content class header</td>
<td>Application-generated message identifier. This identifier will be set as the correlation ID in the response message.</td>
</tr>
<tr>
<td>4</td>
<td>app_id</td>
<td>Content class header</td>
<td>An application identifier for the submitting system instance.</td>
</tr>
<tr>
<td>5</td>
<td>decoder.apiKey</td>
<td>User header</td>
<td>The API key registered to the organization unit.</td>
</tr>
</tbody>
</table>

In addition, the \textit{content_type} content class header must be present in all messages. The value (primary Type, sub type, and parameters) of the mime type is specified against each Message Type.

The message payload is in CSV text format, supporting multiple records. The record delimiter is a line separator. The actual format of the data is specified per message type.

RTI maps provisioned entity identifiers against the combination of the organization identifier and organization unit identifier that is specified in the \textit{user_id} content class header. Messages that contain data must not have a leading organization/organization
unit per line. However, messages that have a filename payload must have this prefix, which must match for each line. Errors are reported for lines with mismatched organization unit IDs.

All provisioning request messages generate a response from the RTI system. All message responses are directed into the organization unit’s response queue.

Determine the outcome of a provisioning request by inspecting the user headers in the response message. The payload contains additional informational detail. The payload content type is an RFC 2046-compliant mime type of text/plain with a content encoding that was used during the request submission. If no content encoding was specified, the default encoding of the deployed RTI system (usually UTF-8) is used to encode the response payload.

The messageType parameter identifies the response message type. The actual value of the parameter is specified in the section detailing the submission message. If the submitted request type cannot be determined, the value of the messageType parameter is PROV.01.GEN.RESPONSE.

Some optional user headers may also appear in the response, and the message consumer may safely ignore them. These headers may be used for troubleshooting and delivery route tracing. Message consumers should not develop any dependency on any of these optional headers.

The following AMQP user headers are common to all responses.

**Table 77. AMQP user headers common to all responses**

<table>
<thead>
<tr>
<th>Number</th>
<th>Name</th>
<th>Message location</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>app_id</td>
<td>Content class header</td>
<td>Unique ID of RTI system runtime.</td>
</tr>
<tr>
<td>2</td>
<td>content_type</td>
<td>Content class header</td>
<td>Fixed text/plain and a messageType parameter.</td>
</tr>
<tr>
<td>3</td>
<td>content_encoding</td>
<td>Content class header</td>
<td>The content encoding used to encode the message payload (same as the encoding used to submit the request). Default: UTF-8</td>
</tr>
<tr>
<td>4</td>
<td>message_id</td>
<td>Content class header</td>
<td>A hash of the submitted payload.</td>
</tr>
<tr>
<td>5</td>
<td>correlation_id</td>
<td>Content class header</td>
<td>Original message ID of the submission request.</td>
</tr>
<tr>
<td>6</td>
<td>timestamp</td>
<td>Content class header</td>
<td>The time in epoch milliseconds when the response was created on the RTI system.</td>
</tr>
<tr>
<td>7</td>
<td>decoder.exchangeRecieved</td>
<td>User header</td>
<td>The original exchange to which the request was submitted.</td>
</tr>
</tbody>
</table>
The following table lists the response codes that may be reported against the `decoder.responseCode` user header.

**Table 78. Response codes**

<table>
<thead>
<tr>
<th>Number</th>
<th>Response codes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>INFO:200</td>
<td>The command ran successfully. Partially successful commands also return this status.</td>
</tr>
<tr>
<td>2</td>
<td>REJECT.SERVER:400</td>
<td>N/A</td>
</tr>
<tr>
<td>3</td>
<td>REJECT.CLIENT:401</td>
<td>If the command includes no header or payload, it is rejected.</td>
</tr>
<tr>
<td>4</td>
<td>REJECT.CLIENT:402</td>
<td>The command was rejected because of an oversized payload (the default maximum size is 10MB).</td>
</tr>
<tr>
<td>5</td>
<td>REJECT.CLIENT:403</td>
<td>If the content type header is not <code>application/text</code> + delimited or <code>application/text+filename</code>, the command is rejected.</td>
</tr>
<tr>
<td>6</td>
<td>REJECT.CLIENT:404</td>
<td>The payload is null or the referenced file cannot be read. The command is rejected.</td>
</tr>
<tr>
<td>7</td>
<td>REJECT.CLIENT:405</td>
<td>This code may have multiple causes: Username or API key is not present in the command headers The API key does not match any existing organization key The user is inactive (effectiveTo property) The command is rejected.</td>
</tr>
<tr>
<td>8</td>
<td>REJECT.CLIENT:406</td>
<td>The user is not an administrator (for a specific range of messages qualified as administration commands). The command is rejected.</td>
</tr>
<tr>
<td>9</td>
<td>REJECT.CLIENT:407</td>
<td>If the content encoding is not supported by the platform, the command is rejected.</td>
</tr>
</tbody>
</table>
An event subscription corresponds to one event stream that is delivered to an OU. To provision an event subscription, submit an AMQP message with a `content_type` header and the following attributes:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Primary type</td>
<td>application</td>
</tr>
<tr>
<td>Sub-type</td>
<td>text+delimited</td>
</tr>
<tr>
<td><code>messageType</code></td>
<td>PROV.01.ESS.SUBMIT</td>
</tr>
<tr>
<td><code>compression</code></td>
<td>true or false (gzip compression is supported)</td>
</tr>
<tr>
<td><code>dataOperation</code></td>
<td>update or delete</td>
</tr>
</tbody>
</table>

When gzip compression is enabled, the `content_type` header sub-type changes to `application/gzip;content=text`.

This attribute is optional and defaults to `update` (event stream subscription parameters are updated when the subscription identifier in a record matches that of an existing subscription).

`delete`: subscriptions that match input subscription identifiers are removed. To delete a subscription, you only need to send the subscription identifier. Other fields in the subscription record are not required.

The values of the `dataOperation` attribute are case-insensitive. For example, `update` and `UPDATE` are both valid.

In the following example, compression is disabled:

```
content_type  "application/text
+delimited;messageType=PROV.01.ESS.SUBMIT;compression=false;
dataOperation=update"
```
The message payload consists of delimited text with one record mapping to one event subscription. The subscription identifier is the only required value per record.

**Table 80. Event subscription message payload**

<table>
<thead>
<tr>
<th>Number</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Subscription identifier</td>
<td>Alphanumeric text: [a-z][A-Z][0-9][-]. Must be unique for an organization/organization unit tuple.</td>
</tr>
<tr>
<td>2</td>
<td>Suspended flag</td>
<td>Text literals (true or false)</td>
</tr>
<tr>
<td>3</td>
<td>Effective from</td>
<td>Date and time format: yyyy-MM-dd’T’HH:mm:ss.SSSZ</td>
</tr>
<tr>
<td></td>
<td></td>
<td>For example: 2013-12-01T23:59:59.999+0000</td>
</tr>
<tr>
<td>4</td>
<td>Effective to</td>
<td>Same as Effective from</td>
</tr>
<tr>
<td>5</td>
<td>Format</td>
<td>Output format name</td>
</tr>
</tbody>
</table>

In the following example, compression is enabled:

```plaintext
content_type "application/gzip;content=text;messageType=PROV.01.ESS.SUBMIT;compression=true;dataOperation=update"
```

Here is an example of a specific message:

```plaintext
RTISUBS001,false,2012-11-23T00:00:00.000+0000,2013-12-01T23:59:59.999+0000
```

A specific message for a delete operation need only contain the subscription identifier:

```plaintext
RTISUBS001
```

Ensure that the ESS message contains the correct output format name. For example, in the following message, the SUB05 subscription is configured to receive the output format named newformat:

```plaintext
SUB05,false,2012-11-23T00:00:00.000+0000,2099-12-01T23:59:59.999+0000,
newformat
```

This type of message is deprecated in RTI Version 1.0. Instead of this message, you can send an ESS message with the appropriate value in the Suspended field. See Event subscription messages.

You can use this type of message to temporarily suspend an active event subscription or reactivate a previously suspended subscription. Submit an AMQP message with the following additional attributes.

---

**Event subscription activation messages (deprecated)**
Table 81. Event subscription activation message attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Primary type</td>
<td>application</td>
</tr>
<tr>
<td>Sub-type</td>
<td>text+delimited</td>
</tr>
<tr>
<td>messageType</td>
<td>PROV.01.SST.SUBMIT</td>
</tr>
<tr>
<td>simulate</td>
<td>true or false Default: false This parameter indicates whether the activation/de-activation should be performed or only simulated. If simulated, a response message indicates what would actually happen as a result of this message.</td>
</tr>
</tbody>
</table>

For example:

```
content_type
"application/text+delimited;messageType=PROV.01.SST.SUBMIT"
```

The message payload is delimited text with one record mapping to one geofence stream activation or deactivation.

Table 82. Event subscription activation message payload

<table>
<thead>
<tr>
<th>Number</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Activation subscription identifiers</td>
<td>Alphanumeric text: [a-zA-Z][0-9]+. Must be unique for an organization/organization unit tuple</td>
</tr>
<tr>
<td>2</td>
<td>De-activation subscription identifiers</td>
<td>As above</td>
</tr>
</tbody>
</table>

The following is an example of two valid messages:

```
ACTIVATE-ME, DEACTIVATE-ME
ACTIVATE_TOO, DEACTIVATE_TOO
```

**Geofence messages**

A geofence is a geographic boundary that conforms to the Open GIS Simple Feature Access standard. RTI supports Version 1.2.1 (see OGC 06-103r4). You can reuse geofences by associating them with different event subscriptions. To provision a geofence, submit an AMQP message with a `content_type` header and the following attributes:

Table 83. Geofence provisioning message attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Primary type</td>
<td>application</td>
</tr>
<tr>
<td>Sub-type</td>
<td>text+delimited</td>
</tr>
<tr>
<td>messageType</td>
<td>PROV.01.GF.SUBMIT</td>
</tr>
<tr>
<td>compression</td>
<td>true or false Optional: gzip compression is supported.</td>
</tr>
<tr>
<td>geomFormat</td>
<td>formatType Optional. Default: wkt (well known text)</td>
</tr>
</tbody>
</table>
Attribute | Value
--- | ---
dataOperation | update or delete
| Optional: defaults to update (ESS parameters are updated when the subscription identifier in a record matches that of an existing subscription).
delete: subscriptions that match input subscription identifiers are removed. To delete a geofence, you only need to send the geofence identifier. Other fields in the geofence record are not required. You cannot delete a geofence that is linked to a subscription (using a linkgeofence). A request to delete a linked geofence returns an error.
The values of the dataOperation attribute are case-insensitive. For example, update and UPDATE are both valid.

For example:

```
content_type "application/text
+delimited;messageType=PROV.01.GF.SUBMIT;geomFormat=wkt;dataOperation=update"
```

The message payload is delimited text with one record mapping to one event geofence. The geofence identifier is the only required value per record.

**Table 84. Geofence provisioning message payload**

<table>
<thead>
<tr>
<th>Number</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Geofence identifier</td>
<td>Alphanumeric text [a-z][A-Z][0-9][]. Must be unique for an organization/organization unit tuple</td>
</tr>
<tr>
<td>2</td>
<td>SRID</td>
<td>The spatial reference identifier. Default: 4326 [WGS84]. Currently, 4326 is assumed, regardless of the value in the message. Other values are ignored.</td>
</tr>
<tr>
<td>3</td>
<td>Buffer</td>
<td>Numeric (integral). A buffer distance around the geometry. The units of the buffer distance must comply with the SRID. For WGS84, this is the &quot;great circle&quot; distance in radians.</td>
</tr>
<tr>
<td>4</td>
<td>Geometry</td>
<td>Well Known Text (WKT) format-encoded geometry entity (Open GIS Simple Feature Access, Version 1.2.1).</td>
</tr>
</tbody>
</table>
| 5      | Spatial Index Name (Optional) | Alphanumeric text [a-z][A-Z][0-9]. Case insensitive. Currently limited to the following values:  
| | | • BufferedRadii90  
| | | • BR90  
| | | • Default Index See Using Selectable Geocoding for more information. |

The following is an example of a specific message:

```
RTISUBS001-GF,4326,0,POLYGON ((-0.1100621103962973 51.61251265985425,  
-0.1488594053073289 51.57421392474571,
```
Chapter 19: Provisioning Manager Message Interface for Loading Reference Data

A specific message for a delete operation need only contain the geofence identifier:

RTISUBS001-GF

The WKT entity supports the use of different spatial reference identifiers (SRIDs). Support for a particular SRID depends on the spatial data available from the access network of the operating entity. At a minimum, the WGS84 reference ellipsoid is typically supported. Therefore, the geometry vertices are expected to be on a matching reference ellipsoid. By default, RTI does not transform coordinates to the supported ellipsoid. Support for such transformations requires a localized implementation (plug-in) for a specific RTI deployment. By default, RTI assumes a WGS84 SRID (4326). The buffer distance must be compatible with the reference system. For WGS84, the buffer is the “great circle” distance in radians.

The messageType parameter in the response message is PROV.01.GF.RESPONSE.

Linkgeofence messages

Associate one or more provisioned geofences with a subscription to activate the subscription against the geographic boundary or boundaries of interest. To provision this association (also known as a linkgeofence), submit an AMQP message with the following attributes in the content_type header:

Table 85. Linkgeofence provisioning message attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Primary type</td>
<td>application</td>
</tr>
<tr>
<td>Sub-type</td>
<td>text+delimited</td>
</tr>
<tr>
<td>messageType</td>
<td>PROV.01.LGF.SUBMIT</td>
</tr>
<tr>
<td>dataOperation</td>
<td>update or delete</td>
</tr>
</tbody>
</table>

Optional: defaults to update. If update is specified, the subscription in the message payload is linked to the geofence(s) in the same message.

delete: This value deletes (or unlinks) an association between the subscription and geofence(s) in the payload.

The values of the dataOperation attribute are case-insensitive. For example, update and UPDATE are both valid.

For example:

content_type “application/text+delimited;messageType=PROV.01.LGF.SUBMIT;dataOperation=update”
The message payload is delimited text with one record mapping to one linkgeofence.

### Table 86. Linkgeofence provisioning message payload

<table>
<thead>
<tr>
<th>Number</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Subscription identifier</td>
<td>Alphanumeric text: [a-z][A-Z][0-9][-]. Must be unique for an organization and organization unit tuple.</td>
</tr>
<tr>
<td>2</td>
<td>Geofence identifiers</td>
<td>Alphanumeric text: [a-z][A-Z][0-9][-]. One or more geofence IDs, separated by semicolons. For example: GF1;GF2;GF3</td>
</tr>
<tr>
<td>3</td>
<td>Reset</td>
<td>Text literal: true or false. Whether existing associations on the subscription should be removed. This setting is ignored when dataOperation is set to delete. Reset applies only to updates (when dataOperation is set to update or is not set).</td>
</tr>
</tbody>
</table>

The subscription identifier and geofence identifier(s) are both required in the linkgeofence message. Here is an example of a specific message:

RTISUBS001,RTISUBS001-GF

For both update and delete messages, define associations between a single subscription and multiple geofences. Geofences must be separated by a semicolon (;). You can also define multiple linkgeofences in a single message payload, using a newline (\n) to separate the mapping of individual subscription IDs to geofence IDs.

For example, an update message payload might contain:

ESS1,GF1;GF2\nESS2,GF2,true

ESS1 is associated with GF1 and GF2. ESS2 is associated with GF2. The reset applies only to ESS2.

### Subscription opt-in messages

Collections of subscribers must be associated with a subscription so that RTI can stream events on those subscribers when the network locates them within the associated geofence(s). A subscriber’s consent to be located and identified in this way is known as a subscriber opt-in.

To associate subscriber opt-ins with event subscriptions, submit an AMQP message with the following attributes.

### Table 87. Subscription opt-in association message attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Primary type</td>
<td>application</td>
</tr>
<tr>
<td>Sub-type</td>
<td>text+delimited</td>
</tr>
<tr>
<td>messageType</td>
<td>PROV.01.SSO.SUBMIT</td>
</tr>
<tr>
<td>Attribute</td>
<td>Value</td>
</tr>
<tr>
<td>------------------</td>
<td>--------------------------------------------</td>
</tr>
<tr>
<td>compression</td>
<td>true or false</td>
</tr>
<tr>
<td></td>
<td>Optional: gzip compression is supported and is useful for large payloads of subscriber IDs.</td>
</tr>
<tr>
<td>translate</td>
<td>true or false</td>
</tr>
<tr>
<td></td>
<td>Optional: true indicates that the customer identifier has to be translated into an IMSI and that the supplied identifier is an alternative synonym for the IMSI. This translation requires RTI integration with an external system to do the translation. Use of the translation capability when no integration has been implemented results in the message being rejected. Default: false</td>
</tr>
<tr>
<td>dataOperation</td>
<td>update or delete</td>
</tr>
<tr>
<td></td>
<td>Optional: update indicates that the subscriber opt-in preferences replace the existing preferences; delete removes the subscriber from the opt-in preferences associated with the corresponding subscriptions. Default: update. The values of the dataOperation attribute are case-insensitive. For example, update and UPDATE are both valid.</td>
</tr>
</tbody>
</table>

For example:

```
content_type "application/text
+delimited;messageType=PROV.01.SSO.SUBMIT;compression=true;translate=false ;dataOperation=update"
```

The message payload is delimited text with one record mapping to one subscriber opt-in against a list of subscriptions.

Table 88. Subscription opt-in association message payload

<table>
<thead>
<tr>
<th>Number</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Subscriber identifier</td>
<td>If translate=true, an MSISDN (phone number). If translate=false, an IMSI key.</td>
</tr>
<tr>
<td>2</td>
<td>Subscription identifiers</td>
<td>Alphanumeric text: [a-z][A-Z][0-9][-]. Must be unique for an organization/organization unit tuple. You can use a semicolon-delimited list (Hex: 3B or Dec: 59) to provision the subscriber against multiple event streams.</td>
</tr>
</tbody>
</table>

Subscription opt-in mapping messages

When collections of subscribers are associated with a subscription, RTI streams events based on the subscriber opt-ins and associated geofences. The subscriber identifier in the event stream is an anonymized identifier combined with the actual subscriber identifier (IMSI or MSISDN). Therefore, event stream consumers need a mapping (association) between the anonymized subscriber identifier and the original subscriber identifier that was submitted during the opt-in registration. This control message provides this mapping.
Chapter 19: Provisioning Manager Message Interface for Loading Reference Data

The control message is sent on the control queue of the organization unit in response to a previously submitted subscriber opt-in message (PROV.01.SSO.SUBMIT). The Provisioning Manager sends these messages whenever a mapping occurs.

Table 89. Provisioning Manager opt-in mapping message attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Primary type</td>
<td>text</td>
</tr>
<tr>
<td>Sub-type</td>
<td>plain</td>
</tr>
<tr>
<td>messageType</td>
<td>PROV.01.SSO.CONTROL</td>
</tr>
<tr>
<td>compression</td>
<td>true or false Default: true</td>
</tr>
</tbody>
</table>

The message sent by the Provisioning Manager is delimited text with one header record that contains an informational status line (which can be ignored), followed by a new line. Subsequent records are delimited fields that contain the following information.

Table 90. Provisioning Manager opt-in mapping message payload

<table>
<thead>
<tr>
<th>Number</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Subscriber id</td>
<td>The original supplied subscriber identifier: an IMSI or MSISDN (if title translation was requested)</td>
</tr>
<tr>
<td>2</td>
<td>Hashed identifier</td>
<td>Alphanumeric text [a-z][A-Z][0-9].</td>
</tr>
</tbody>
</table>

Submit a stream subscription mapping message to explicitly request anonymized subscriber identifiers to be mapped to the actual subscriber identifiers.

Table 91. Stream subscription mapping request attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Primary type</td>
<td>application</td>
</tr>
<tr>
<td>Sub-type</td>
<td>text+delimited</td>
</tr>
<tr>
<td>messageType</td>
<td>PROV.01.SSM.SUBMIT</td>
</tr>
</tbody>
</table>

For example:

```plaintext
content_type "application/text +delimited;messageType=PROV.01.SSM.SUBMIT;translate=true;compression=false"
```

The message payload is delimited text with one record mapping to a set of subscriber identifiers that have the same translation value.

Table 92. Stream subscription mapping request payload

<table>
<thead>
<tr>
<th>Number</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Subscription identifiers</td>
<td>Alphanumeric text [a-z][A-Z][0-9]. Must be unique for an organization/organization unit tuple. You can specify multiple identifiers in a semicolon-delimited list.</td>
</tr>
</tbody>
</table>
For example, here are two SSM messages:

```
SUB05,true
SUB06;SUB07,true
```
This chapter presents the following topic:

Output format for event streams.................................................................349
Output format for event streams

This section describes the output format, or distribution format, of messages or files sent to clients to satisfy event stream subscriptions. RTI produces two event streams for every subscription: a location stream and a transaction stream. Location events contain a subset of the information that is produced for transaction events. The location stream targets use cases in which the client is more interested in the subscriber location information for an event. The transaction stream targets uses cases in which the client is more interested in what happened on the network during a transaction.

The output is distributed with a payload that has a content-type of text/csv (plain text, comma-separated values). Each CSV record is separated by a new line. The following sections show the position and content of each field in each output record.

If the following property is set to false, the date field values are in milliseconds instead of UNIX epoch time:

rti.distgrid.format.transaction.eventTimeAsUnixEpoch

Transaction events Each transaction event in CSV format contains the following fields.

The contents of the output vary, based on configuration properties that enable or disable certain fields. Therefore, the following position numbers may not be accurate for your output.

Table 93. Transaction event CSV fields

<table>
<thead>
<tr>
<th>Position</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>IMSI</td>
<td>Unique identifier for the network subscriber. Plain text or anonymized 32-digit hash value, depending on system settings.</td>
</tr>
<tr>
<td>2</td>
<td>Subscriber MCCMNC</td>
<td>A code that uniquely identifies the country and the carrier of the subscriber.</td>
</tr>
<tr>
<td>3</td>
<td>MSISDN</td>
<td>Subscriber’s MSISDN. Plain text or hash value, depending on system settings.</td>
</tr>
<tr>
<td>4</td>
<td>IMEI</td>
<td>International Mobile Station Equipment Identity key; serial number suppressed to zero. Plain text or hash value, depending on system settings.</td>
</tr>
<tr>
<td>5</td>
<td>Called</td>
<td>The number that was called for voice calls, or the Access Point Network (APN) that was accessed for data transactions. Plain text or anonymized 32-digit hash value, depending on system settings.</td>
</tr>
<tr>
<td>6</td>
<td>Start date/time</td>
<td>When the transaction started, as detected by the network. By default, this field is in UNIX epoch time (the number of seconds since January 1, 1970.)</td>
</tr>
<tr>
<td>7</td>
<td>End date/time</td>
<td>When the call ended, as detected by the network. By default, this field is in UNIX epoch time (the number of seconds since January 1, 1970.)</td>
</tr>
<tr>
<td>8</td>
<td>First LAC or TAC</td>
<td>Initial transaction location in terms of its location area code (LAC for 2G and 3G transactions) or Tracking Area Code (TAC) for 4G transactions. A transaction may consist of a sequence of network events; the first LAC or TAC records where the network initially located the caller.</td>
</tr>
</tbody>
</table>
### Position | Name | Description
--- | --- | ---
9 | First cell tower ID or SAC or ECI | Initial transaction location in terms of its cell tower ID (for 2G) or ECI (EU Tran Cell ID) for 4G, or Service Area Code (SAC) for 3G transactions. A transaction may consist of a sequence of network events; the first cell ID or ECI or SAC records where the network initially located the caller.

Taken together, fields 8 and 9 (and 10 and 11) form location tuples. You can configure the system to use geo-coordinate values for this tuple instead: latitude and longitude values instead of LAC/TAC and cell ID/SAC values:

- `rti.distgrid.format.transaction.geoCoordinatesMode`

10 | Current LAC or TAC | Same as field 8, except that this location defines where the last event in the transaction took place.

11 | Current cell tower ID or SAC | Same as field 9, except that this location defines where the last event in the transaction took place.

12 | Radii90 | This field represents the 90% probability around the cell tower centroid, as stored in the cell tower reference data. If the `rti.distgrid.format.transaction.geoCoordinatesRadii90` property is set to `true`, this field is inserted into the output stream as field 12 and the remaining field positions are incremented by 1. If this property is `false`, the protocol field is in position 12.

13 | Protocol (data record type) | Number that identifies the source protocol for the transaction:

- 2=A (voice)
- 3=Gb (data)
- 5=iucs (3G voice) or iups (3G data)
- 7=radius
- 9=s1mme
- 10=gngi

14 | Tag (bit mask) | 16-bit value. Enrichment information is provided by RTI.

- Bit 15 is set for subscriber intercept, bit 14 for device intercept, and bit 13 for location intercept.
- Bit 1 is set for subscriber trace, bit 2 for device trace, and bit 3 for location trace.
- The other bits are currently not set.

This field may not be present in the output, depending on the setting of the `rti.distgrid.format.transaction.extTagInfo` property. By default, this property is set to `true`, which disables the field. Therefore the position of the following fields may be decremented by 1. For example, the status field will be field 14.

**Note:** Field 14 is deprecated. Complete the instructions in Configuring Output Streams to configure output streams.
### Output Format for Event Streams

#### Chapter 20

**EMC Real-Time Intelligence (RTI) 3.1.2 Administrators Guide**

<table>
<thead>
<tr>
<th>Position</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>Status</td>
<td>Bit mask of network protocol status bits that can be unpacked to describe what happened in the transaction. The bit mask conforms to the Output Hybrid Data Record (OHDR) specification for the OHDR version in use. If the extended transaction format is enabled (rti.distgrid.format.transaction.extendedTranFormat is true), additional protocol-specific information is embedded in this field. See the &quot;Transaction Status&quot; description following this table.</td>
</tr>
<tr>
<td>16</td>
<td>RTI ingest time</td>
<td>When RTI first received the transaction event from the network, in UNIX epoch time (the number of seconds since January 1, 1970.)</td>
</tr>
<tr>
<td>17</td>
<td>RTI processed time</td>
<td>When the transaction event became available to the RTI system in the ingest grid, in UNIX epoch time (the number of seconds since January 1, 1970.)</td>
</tr>
<tr>
<td>18</td>
<td>RTI export time</td>
<td>When the transaction event left the distribution grid, in UNIX epoch time (the number of seconds since January 1, 1970.)</td>
</tr>
<tr>
<td>19</td>
<td>Not used</td>
<td>This field is currently not used, but lines in the output contain a trailing &quot;,,&quot; to indicate that the last column in the row is empty.</td>
</tr>
</tbody>
</table>

#### Transaction status information

The contents of the status field for transaction events contain extended protocol-specific information when the rti.distgrid.format.transaction.extendedTranFormat property is set to true. The extended details are embedded inside the status field, in the positions shown in the following table.

Note that the information provided depends on the protocol. For details about this information, refer to the standard 3GPP specifications and the network protocol specifications.

If you are creating an event subscription, you cannot request the extended format explicitly; you must ensure that the RTI system administrator makes the extended format available.

If the rti.distgrid.format.transaction.decodeStatusBits property is set to true, the Status Bits and Timeout Bits information is further defined.

#### Table 94. Transaction status information

<table>
<thead>
<tr>
<th>Field</th>
<th>Field description (per protocol)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Adr                      Gb                  IuCS</td>
</tr>
<tr>
<td>1</td>
<td>Status Bits               Transaction status</td>
</tr>
<tr>
<td>2</td>
<td>TransactionType           acctOutputOctets</td>
</tr>
<tr>
<td>3</td>
<td>CallType                  acctInputPackets</td>
</tr>
<tr>
<td>4</td>
<td>Timeout Bits              acctOutputPackets</td>
</tr>
<tr>
<td>5</td>
<td>BSS MAP Cause             GMM SM Cause</td>
</tr>
</tbody>
</table>
### Location events

Each location event in CSV format contains the following fields.

<table>
<thead>
<tr>
<th>Position</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>IMSI</td>
<td>Unique identifier for the network subscriber. Plain text or anonymized 32-digit hash value, depending on system settings.</td>
</tr>
<tr>
<td>2</td>
<td>Subscriber MCCMNC</td>
<td>A code that uniquely identifies the country and the carrier of the subscriber.</td>
</tr>
<tr>
<td>3</td>
<td>IMEI</td>
<td>International Mobile Station Equipment Identity key; serial number suppressed to zero. Plain text or hash value, depending on system settings.</td>
</tr>
<tr>
<td>4</td>
<td>MSISDN</td>
<td>Subscriber's MSISDN. Plain text or hash value, depending on system settings.</td>
</tr>
<tr>
<td>5</td>
<td>End date/time</td>
<td>When the call ended, as detected by the network. By default, this field is in UNIX epoch time (the number of seconds since January 1, 1970.).</td>
</tr>
<tr>
<td>6</td>
<td>Current LAC/TAC</td>
<td>Transaction location in terms of its location area code (LAC for 2G and 3G transactions) or Tracking Area Code (TAC) for 4G transactions. A transaction may consist of a sequence of network events; the current LAC or TAC records where the network last located the caller.</td>
</tr>
<tr>
<td>7</td>
<td>Current cell tower ID or SAC</td>
<td>Transaction location in terms of its cell tower ID (for 2G and 4G transactions) or (SAC for 3G transactions). A transaction may consist of a sequence of network events; the current cell ID or SAC records where the network last located the caller. Taken together, fields 6 and 7 form location tuples. You can configure the system to use geo-coordinate values for this tuple instead: latitude and longitude values instead of LAC/TAC and cell ID/SAC values: rti.distgrid.format.location.geoCoordinatesMode</td>
</tr>
<tr>
<td>8</td>
<td>Radii</td>
<td>Subscriber's estimated distance in meters from the cell tower. RTI selects a 70%, 80%, or 90% confidence factor and returns one distance value based on this choice. For example, RTI may be 80% confident that the subscriber is 500 meters from the cell tower, within a sector or circle served by that tower.</td>
</tr>
</tbody>
</table>
### Output Format for Event Streams

<table>
<thead>
<tr>
<th>Position</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Number that identifies the source protocol for the transaction: 2=A (voice) 3=Gb (data)</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>Protocol (data record type)</td>
<td>Number that identifies the source protocol for the transaction: 2=A (voice) 3=Gb (data) 5=iucs (3G voice) or iups (3G data) 7=radius 9=s1mme 10=gngi</td>
</tr>
<tr>
<td>10</td>
<td>Status bits</td>
<td>Bit mask of network protocol status bits that can be unpacked to provide information about the location event.</td>
</tr>
</tbody>
</table>

**Full format output**

The LORS output stream can be formatted using the full format. The attributes displayed in this format are in the following sequence:

`protocolName, imsi, imei, mccmnc, msisdn, startTimeUTC, timeUTC, ingestTime, processedTime, lac, cellTower, latitude, longitude, firstLac, firstCellTower, radii90, transactionTarget, eventType, eventStatus, mobileEquipment, subscriberProfile, protocolDetails, protocolAttributes`

**Format details**

- Attributes are separated by a comma ",".
- Null values, empty strings (length zero), and blank spaces are not displayed. The output shows two commas (,,) where the attribute would have been if present.
- `mobileEquipment`, `subscriberProfile`, `protocolDetails`, and `protocolAttributes` can be hidden from generated output by using configuration, which is described in the section below.
- MobileEquipment is a composite field with the following sub-fields:
  
  `{deviceGeneration, model, manufacturer, tac, uaProfile}`.

  The {} are delimiters for the set of sub-fields from the rest of the output.

  MobileEquipment is added to the event as part of enrichment from RTI reference data. If no value is found to enrich the record, then a default value is added which is:

  `{UNKNOWN, NA, NA, NA, NA}`

**Note:** For full format, unlike other formats, `mobileEquipment` appears only outside the `protocolDetails` section.

- SubscriberProfile is a composite field with the following sub-attributes:

  `{externalId, hashedMsisdn, homePostCode [group1, group2, ... groupN]}`.
Chapter 20: Output Format for Event Streams

The { } are delimiters for the set of sub-fields from the rest of the output.
Within the { }, the list of subscriber groups is displayed as a comma separated enclosed list [group1,group2,...,groupN].

SubscriberProfile is added to the event as part of enrichment from RTI reference data. If no value is found to enrich the record, then a default value is added which is {NA,NA,NA[}.

Note: For Full Format, unlike other formats, subscriberProfile appears only outside the protocolDetails section.

- protocolDetails is a composite field with key and value pairs of attributes decoded by the protocol adapters as well as attributes enriched from RTI reference data. Refer to the data dictionary for the details of attributes decoded for each protocol. The format is as follows:

  [key1=value;key2="stringValue";key3=value,...,keyN=value]

  where [ ] are delimiters for the set of subs-fields from the rest of the output. Double quotation marks are used around values of type String. For all other types, quotation marks are not used.

  Within the [], null values are not displayed at all. Empty strings (length zero) and blank spaces are displayed as key="" (double quotes with nothing in between).

- protocolAttributes is a composite field with key and value pairs of attributes added by RTI. The format details for protocolAttributes is exactly the same as protocolDetails.

Configuring full format
To enable this format, set the property rti.distgrid.lors.format to the value “full” on each distribution grid instance using RTISH as shown below.

    rti set property --instances distgrid-two@* --name rti.distgrid.lors.format --value full

Hiding composite fields
The following properties can be set to hide composite fields from appearing in the output.

Table 96. Configuration properties for hiding composite fields

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>rti.distgrid.format.full.includeMobileEquipment</td>
<td>Includes or hides mobileEquipment</td>
<td>true (includes)</td>
</tr>
<tr>
<td>rti.distgrid.format.full.includeSubscriberProfile</td>
<td>Includes or hides subscriberProfile</td>
<td>true (includes)</td>
</tr>
<tr>
<td>rti.distgrid.format.full.includeDetails</td>
<td>Includes or hides protocolDetails</td>
<td>true (includes)</td>
</tr>
<tr>
<td>rti.distgrid.format.full.includeAttributes</td>
<td>Includes or hides protocolAttributes</td>
<td>true (includes)</td>
</tr>
</tbody>
</table>
For example, to hide the mobileEquipment and subscriberProfile, one should set as follows:

```
rti set property --instances distgrid-two@* --name rti.distgrid.format.full.includeMobileEquipment --value false
rti set property --instances distgrid-two@* --name rti.distgrid.format.full.includeSubscriberProfile --value false
```

### Fields in ProtocolDetails

The protocolDetails field is a composite one with key and value pairs of attributes decoded by the protocol adapters as well as attributes enriched from RTI reference data. Refer to the data dictionary for the details of attributes decoded for each protocol. The following attributes are added to the protocolDetails field as part of reference data enrichment by RTI.

Table 97. ProtocolDetails attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>eventStartTime</td>
<td>Present only if a value is added by protocol adapter</td>
</tr>
<tr>
<td>eventEndTime</td>
<td>Present only if a value is added by protocol adapter</td>
</tr>
<tr>
<td>mobileEquipment</td>
<td>If no value is found during enrichment, it adds the default {UNKNOWN, NA, NA, NA, NA}</td>
</tr>
<tr>
<td>subscriberProfile</td>
<td>If no value is found during enrichment, it adds the default {NA, NA, NA}.</td>
</tr>
<tr>
<td>latitude</td>
<td>Present only if a value is found during enrichment</td>
</tr>
<tr>
<td>longitude</td>
<td>Present only if a value is found during enrichment</td>
</tr>
<tr>
<td>radii90</td>
<td>Present only if a value is found during enrichment</td>
</tr>
<tr>
<td>cellType</td>
<td>Present only if a value is found during enrichment</td>
</tr>
<tr>
<td>locationProvider</td>
<td>Present only if a value is found during enrichment</td>
</tr>
<tr>
<td>processedTime</td>
<td>Present only if a value is found during enrichment</td>
</tr>
</tbody>
</table>
Chapter 21 Setting up a Grafana Dashboard

This chapter presents the following topic:

Setting up a Grafana dashboard ................................................................. 357
Setting up a Grafana dashboard

A Grafana dashboard provides a visual graph of the messages coming out of RTI. Users can setup an InfluxDB transport to take messages coming out of the AMQP server, transform the message into an InfluxDB format, and save them to InfluxDB. When the data is in InfluxDB, Grafana can generate charts to visualize it.

**Note:** The Grafana and InfluxDB products are not supported by EMC. Those products have been used to show an example of how to visualize the RTI output.

Follow these basic instructions to install Grafana:

1. **Install InfluxDB:**

   ```bash
   yum install https://dl.influxdata.com/influxdb/releases/influxdb-0.9.6-1.x86_64.rpm
   ```

2. **Start InfluxDB as a service:**

   ```bash
   systemctl enable influxdb
   ```

3. **Create the InfluxDB database:**

   ```bash
   influx
   ```

   ```bash
   create database rti with duration 1d
   ```

   Creates a database named rti, that expires data older than 1 day.

4. **Install Grafana:**

   ```bash
   yum install https://grafanarel.s3.amazonaws.com/builds/grafana-3.0.2-1463383025.x86_64.rpm
   ```

5. **Start Grafana as a service:**

   ```bash
   systemctl daemon-reload
   systemctl enable grafana-server.service
   systemctl start grafana-server.service
   ```

   You should be able to browse to Grafana at localhost:3000.

6. **Install Grafana plugins.**

   Any plugins must go in the /var/lib/grafana/plugins directory.

After Grafana is up and running, connect to the InfluxDB database and create charts for the desired metrics. For more information, refer to the Grafana documentation.
This appendix presents the following topics:

Glossary .................................................................................................................. 359
Glossary

**AMQP**
Advanced Message Queuing Protocol—Messaging protocol supported by RabbitMQ and used by RTI to stream events to consumers based on subscriptions.

**Blacklist - opt-in**
A Blacklist opt-in is at the top of the opt-in hierarchy and a peer to Global opt-ins. Blacklisted subscribers and locations are disqualified from processing early in the event processing cycle.

**Cell tower**
Cellular network tower that is the source of raw events received by the RTI ingester (using a third-party probe).

**CocoaRest**
Client UI for working with REST interfaces.

**Codec**
A software program that decodes a stream of information that is based on the protocol in which it is coded. Mobile network events must be decoded on a per-protocol basis.

**Distribution Grid**
The GemFire grid that receives events from the Ingest Grid and distributes them to consumers by using different queues.

**DR opt-in**
Data record opt-in—identifies raw events that do not leave the ingester grid (no further processing).

**Exchange name**
Combination of organization name and organization unit name, forming a specific link that allows event subscriptions to be submitted and consumed by third parties.

**External opt-in**
Opt-ins for locations (geofences) and subscribers identified by external (third-party) event subscriptions in Provisioning Manager messages.

**Femtocell**
A small cellular base station that is designed for use in a home or small business and connected to the core network over a local IP connection. A femtocell enables service providers to extend service coverage to homes and businesses where access would otherwise be limited or unavailable.
Appendix A: Glossary

**Gateway**
A high-performance connection between the two GemFire grids in the RTI system. A GemFire gateway defines a single remote distributed system site in a multi-site installation and manages communication with the remote site.

**GemFire**
Distributed system software that provides in-memory parallel processing of large amounts of data.

**Geofence**
The definition of a geographical area in terms of a set of coordinates.

**Global opt-in**
Applied in the ingest grid for early filtering of events that are based on subscribers and locations.

**Grid**
See ingest grid and distribution grid.

**Grid 1**
Alternative name for the ingest grid.

**Grid 2**
Alternative name for the distribution grid.

**IMEI**
International Mobile Station Equipment Identity—a number that identifies a mobile phone, but has no connection to the phone subscriber. The IMEI identifies valid devices and can be used to blacklist a stolen phone, for example, to render the phone useless on the network.

**IMSI**
International Mobile Subscriber Identity—The SIM card ID that identifies a phone subscriber. This number is tracked through RTI but hashed in the output for event subscriptions.

**Ingester**
The RTI component that receives raw events from cell towers, applies some initial filters, and sends them to the ingest grid.

**Ingest Grid**
The GemFire grid that receives events from the Ingester, applies some filters, and sends them to the Distribution Grid.

**JMX Java Management Extensions**
Java Management Extensions—a Java technology that provides tools for managing and monitoring applications, system objects, devices, and networks. These resources are represented by objects called MBeans.

**Linkgeofence**
An association between a subscription and one or more geofences.
**LAC**

Location Area Code—A unique 16-bit number assigned to a location area, which is a group of base stations. The location area code is broadcast by each base station.

**LANDS**

Location AND Subscriber.

**Locator**

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.

**LOGBack**

Logging system used by RTI.

**LORS**

Location OR Subscriber.

**MBean**

A Java managed object or resource. In the context of RTI, you can view specific MBean attributes and values to monitor the system, such as counts of events flowing through the grids.

**MSISDN**

Mobile Station International Subscriber Directory Number—A number uniquely identifying a subscription in a mobile network: the telephone number to the SIM card in a cellular phone.

**Opt-in**

Consent on the part of a private party (or a corporate contract) to be tracked for advertising purposes and receive resulting promotions, email messages, or other solicitations. In RTI, subscribers (mobile device users) and locations (geographic areas) can be opted into event streams.

**OU**

Organization unit—An entity that can submit event subscriptions to RTI. An OU is a unique identifier that consists of an organization name and a unit ID.

**Probe**

A third-party device that collects raw data from network cell towers.

**Properties**

Configuration parameters or switches that enable, disable, or set runtime values for RTI components.

**Protocol**

A set of standard rules for representing and handling data sent over a communications channel. RTI supports several different telecommunications protocols.
Appendix A: Glossary

**Provisioning Manager**
RTI component that communicates with RabbitMQ and the distribution grid to relay messages for event subscriptions.

**RabbitMQ**
Messaging software that serves as the AMQP broker for the RTI system.

**RDL**
Reference Data Loader—Command-line interface that loads CSV files into the RTI system for reference during event processing.

**Reference data**
Data loaded into the grids that enriches and filters events as they are processed through the system.

**Region**
GemFire storage area used to contain reference data.

**REST interface**
An interface that discovers resources in a distributed system and exposes them to users by using browser links (URIs). The RTI REST interface is used to view data in the GemFire regions. (REST stands for REpresentational State Transfer.)

**Routing key**
The combination of the first two fields in OU records identify both the orgUnit value and the exchange value in event subscription messages, and are used as the prefix for the routingKey and responseQName:

```plaintext
ext.orgUnit = "BRN_DG"
ext.apiKey = "6d051b96f732e6607b30f9b9d5caf39a"
ext.exchange= "BRN.DG"
ext.routingKey= "BRN.DG.request"
ext.responseQName= "BRN.DG.response"
```

**Subscriber**
A mobile phone user who might be the target of a campaign that is implemented by subscribing to events processed through RTI.

**Subscription**
A request from an external consumer to receive a flow of events that meet certain criteria. Also known as an event stream subscription (ESS).

**Syphon**
Utility that captures events in JSON format output files before the events leave the ingest grid.

**TMSI**
Temporary Mobile Subscriber Identity—a frequently changing subscriber identity that is typically transmitted between the mobile device and the network. The global IMSI is rarely sent over the mobile network. RTI maps TMSI values to IMSI values.
**Tracing**
Tagging of events that involve specific devices, subscribers, and locations as they travel through the RTI system. These events are logged.

**Type Allocation Code**
Type Allocation Code—Initial eight-digit portion of the 15-digit IMEI code that uniquely identifies wireless devices.

**VisualVM**
Software that you can use to visualize RTI metrics