VMware Smart Assurance System Administration Guide

VMware Smart Assurance 10.0.0

vmware®

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

https://docs.vmware.com/

If you have comments about this documentation, submit your feedback to

docfeedback@vmware.com

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

Copyright © 2019 VMware, Inc. All rights reserved. Copyright and trademark information.

Contents

1 Introduction 5

Setting the Locale 5

Locating and modifying SDK files 5

Licensing SDK software 6

Controlling the startup of SDK programs 6

Securing access to SDK software 6

Operation of the Broker 6

Managing log files 7

2 Locale 8

What is a locale 8

Locale types 9

Standards 10

3 Locating and Modifying VMware Smart Assurance Files 12

VMware Smart Assurance directories 12

User-modifiable files 13

How SDK software locates user-modifiable files 16

Where SDK software writes output files 16

4 Controlling the Startup of VMware Smart Assurance Software 20

Root access requirement on Unix and Linux 20

Overview of services 21

About the sm_service utility 21

sm_serviced and ic-serviced 22

The sm_service command line 23

Installing programs as services with sm_service 25

Displaying installed services and their status 26

Starting and stopping services with sm_service 28

Modifying service parameters with sm_service 29

Removing services with sm_service 30

Default parameters for VMware services 30

Starting a Domain Manager 34

Starting other DMT processes 35

Running as non root on UNIX and Linux systems 37

5 Securing Access to VMware Software 40

SDK security features 40

How SDK security works 41
Configuring authentication 44
Controlling authentication 55
Encryption 56
Encrypted passwords 58
Encrypted seed files 59
Encrypted connections 60
Configuring a secure Broker 63
Additional safeguards 64
Example security configurations 65

6 Operation of the Broker 68
Overview of the Broker 68
Viewing the registry of the Broker 69
How the SDK clients find the Broker 69
How to change the Broker environment variable 70
Securing the Broker 72

7 Managing Log Files 73
Overview of message logging 73
Retrieving and setting log, error and trace levels at runtime 78
Log files 79
Starting a new log file 83
Controlling the number of saved log files 84

8 Environment Variables Used by VMware Software 86
How variable values are interpreted 86
Methods for setting environment variables for VMware software 87
Environment variables for users 89

9 Wildcards Used by VMware Software 105
Types of wildcard patterns 105

Introduction

1

This chapter includes the following topics:

- Setting the Locale
- Locating and modifying SDK files
- Licensing SDK software
- Controlling the startup of SDK programs
- Securing access to SDK software
- Operation of the Broker
- Managing log files

Setting the Locale

A locale is a combination of language and country identifiers, and the specification of the locale is important for the purpose of SDK externalized textual data. Clients and Domain Managers can be running in different locales but it is important that textual data transferred to a client as part of error handling by the Domain Manager appear in the language and country of the client. Chapter 2 Locale describes how to specify the locales.

Locating and modifying SDK files

Some SDK files can be modified to meet the needs of your network environment. Chapter 3 Locating and Modifying VMware Smart Assurance Files describes the directory structure of the SDK software. The chapter also discusses the files that can be modified and how to locate the files and provides procedures for modifying those files. Additionally, the chapter describes the files created by SDK software (for example, logs, repositories, and saved consoles), and details how to control their location.

Licensing SDK software

SDK software requires a license file to function. To evaluate the software, you can obtain a time-limited evaluation license that provides access to specified features. After you purchase the software, you will need to set up a permanent license file. Chapter 4, “License Administration,” provides procedures for obtaining both evaluation and permanent license files for your SDK software. Topics include:

- Setting up the license file
- Starting the FLEXnet license server
- Pointing the SDK software to the license server
- Licensing additional SDK programs and adding licenses

Controlling the startup of SDK programs

SDK programs are started can either be installed as services or as manual processes. The section Chapter 4 Controlling the Startup of VMware Smart Assurance Software provides procedures that use the sm_service utility to:

- Install programs as services
- Start and stop servicesSDK programs
- Start and stop processes
- Modify the runtime parameters of servicesSDK programs

Securing access to SDK software

Security is a critical concern in the world of large-scale distributed networks. Therefore, SDK security provides a means by which network administrators can set up security to control access to the software. Chapter 5 Securing Access to VMware Software describes how SDK security works, as well as the authentication processes used by the software. It then provides procedures for setting up security and controlling access to SDK programs. The chapter also discusses how to set up encrypted connections between SDK clients and servers.

Operation of the Broker

SDK client programs, such as a console or an adapter, use the Broker to determine where Managers are running. When a Manager starts, it registers the hostname of the machine it is running on, as well as the TCP port it is listening to, with the Broker. Thereafter, SDK clients retrieve this information from the Broker so that they can communicate with the Manager. Chapter 6 Operation of the Broker describes the functions of the Broker, provides procedures for viewing the Broker’s registry, and explains how to change the Broker’s environment variables.

Managing log files

Log files grow indefinitely, though slowly, under normal conditions, and need to be managed. In addition, SDK software can maintain up to 1,000 backup log files of each log. For example, if you are generating two log files, MYDM_en_US_UTF-8.log and MYDM_fr_UTF-8.log file, you can maintain up to 1,000 backup log files for the MYDM_en_US_UTF-8.log and 1,000 backup log files for the MYDM_fr_UTF-8.log.

You can configure the number of log files retained by SDK software. Chapter 7 Managing Log Files discusses the management of log files, and describes how to “roll over” the files.

Locale

2

This chapter includes the following topics:

- What is a locale
- Locale types
- Standards

What is a locale

A *locale* is a set of parameters that defines:

1 Language

2 Country (or region)

3 Other specific cultural elements that vary

For the purpose of SDK externalized textual data, the language and country data of the locale specification are important.

Languages are specified in two lowercase ASCII letters. Some examples are en for English and fr for French.

Country identifiers are specified as two or three uppercase ASCII letters. Some examples are: US, CA, FR, and JP, for United States, Canada, France, and Japan, respectively.

Standards on page 19 provides more information on language codes and country identifiers.

A locale is a combination of the language and country identifiers into one string, connected by the underscore (_) character. For example, the locale for English (en) as spoken in the United States (US) is defined as en_US.

Client and Domain Manager use different locales

Client programs may be running in a locale that is not the same as the locale of the Domain Manager. For example, a Domain Manager may be running in a process where the locale is specified as “French as used in France.” Different end-users of an application may be concurrently connected, and use different locales, for example “German as spoken in Germany,” and “Spanish as spoken in Spain,” or “German as spoken in Austria.”

It is important that textual data transferred to the client as part of error handling by the Domain Manager appear in the language and country of the client. A German client will see German exception messages, even if the client application is connected to a Domain Manager whose preferred language (for logging) is French.

The server will respect the locale of the client when rendering several types of textual output. Applications may also use classes and methods in the clsapi subsystem to enable locale-aware operations, where appropriate.

Locale types

Each process' locale is determined by the SM_LOCALE environment variable, as seen by that process. When a client process attaches to a server process, they negotiate on the locale for that session. The SM_LOCALE environment variable is discussed in SM_LOCALE.

Process locale

You can specify the locale of any SDK program, such as a client, Broker, server, utility by using the SM_LOCALE environment variable. For all such programs, the SDK initialization logic validates the specified value, and ensures that all internationalization features can use it predictably.

The locale can also be specified for Java applications. Use the Java property com.vmware.cmp.dmt.locale on the startup of the Java Virtual Machine or on the command line of any SDK program that supports the -java command operand.

Client locale

The locale of a client program may be different from the locale of the Domain Manager. There are two different types of client locales:

- Client process locale.
- Client session locale.

  The client process locale and the client session locale do not need to be the same.

Client process locale

The client process locale is the locale that is set by the SM_LOCALE variable in the environment of the client program, such as **dmctl**, **brcontrol**, and others. The client process locale is used by default when rendering messages generated in the client process.

Client session locale

The client session locale is the locale that is sent to a Broker or Domain Manager in two possibly ways:

- The client process locale is by default sent to the Domain Manager, and identifies the initial session locale.

- The client session locale can be changed by calling the setLocale() or setRemoteLocale() method. Calling one of these methods causes a protocol verb to be executed at the server, which causes the server to remember the client session locale.

  The client program may use the setLocale() client API calls to indicate to the Domain Manager a select locale. Setting the locale that the Domain Manager remembers for the client does not change the configured setting of the client process locale.

Domain Manager process locale

The Domain Manager process locale is the one set by the SM_LOCALE variable in the environment of the Domain Manager. The Domain Manager locale primarily influences logging output, and does not impact the processing of locale for client processes or the client sessions. Conversely, the client locale, configured or set by API call, does not influence the locale handling for the Domain Manager output.

Support locale

The process locale specified by SM_LOCALE and the Java property specifies the language for output that the user sees. However, VMware engineering and support teams must be able to interpret log data and other output in English. SDK programs, primarily server processes, will always generate diagnostic information in the support locale, which is en_US. The main impact of the difference between the process locale and the support locale applies to the SDK logging system. See Overview of message logging for more details.

Standards

Several internationalization standards apply directly to SDK applications.

Unicode

Unicode is the universal standard of encoding for written characters and text, covering all known world languages. UTF-8 (8-bit form) is used by SDK applications.

ISO 639 language codes

Languages are specified in two lowercase ASCII letters. Some examples of standard codes for language are: en, fr, and ja for English, French, and Japanese respectively.

ISO 3166 country codes

This standard defines the universally recognized codes for county (or region). There are two alphabetic standards for country codes:

- ISO-3166-1 Alpha-2 describes the two-letter codes.

- ISO-3166-1 Alpha-3 describes the three-letter codes.

  Country identifiers are specified as two or three uppercase ASCII letters. For example, US for United States and JP for Japan.

  Language and country identifiers are combined into one string, connected by the underscore character. For example, en_US for English as used in the US, and ja_JP for Japanese as used in Japan.

Locating and Modifying VMware Smart Assurance Files

3

This chapter includes the following topics:

- VMware Smart Assurance directories
- User-modifiable files
- How SDK software locates user-modifiable files
- Where SDK software writes output files

VMware Smart Assurance directories

ITOps directories lists the SDK directories and a brief description of their contents

Table 3-1. SDK directories

Directory	Contents
/bin	Executables for starting and interacting with Domain Managers as well as utilities.
/classes	SDK jar files and third-party jar files upon which SDK programs depend.
/conf	Domain Manager configuration files.
/doc	User and reference guides.
/examples	Sample files for C, C++, and Java APIs as well as examples of MODEL applications and JIIM.
/include	The .h and .mdl files for components of the SDK programs.
/integration	Sample probe for the integration accessor.
/l10n	The l10n directory name is a lowercase L, followed by the number 10 and then a lowercase n. Contains Resource Packages (which hold externalized and localized data, such as error strings).
/lib	Shared libraries.
/local	Top-level directory where local copies of configuration, rules, and script files are created and stored. The default location where files such as repository and log files are written. The/local/l10n directory contains new Resource Packages introduced by update kits, patches, language packs, and so on.
/perl	Files used for the Perl API.

Table 3-1. SDK directories (Continued)

Directory	Contents
/rules	ASL rule sets.
/script	Utility scripts such as the license server installation scripts.

In addition to the common subdirectories, the VMware Smart Assurance Service Assurance Manager (/SAM) includes two additional subdirectories: /actions (client and server) and /consoles.

User-modifiable files

The SDK installation includes system files, configuration files, ASL rule set files, scripts, templates, as well as third-party software files that the SDK software uses.

Never alter SDK system files, such as executables, shared libraries, MODEL files, and setup files. Do not alter third-party software system files provided with the Java Runtime Environment.

As part of the configuration process, you will need to make changes to some of the user-modifiable files. The following are examples of user-modifiable files:

- Configuration files, such as ics.conf, clientConnect.conf, and serverConnect.conf (you are required to edit these)
- ASL rule sets, such as syslog_mgr.asl
- Sample actions, such as ics-ping and ics-telnet
- Template files used to import data, such as service.data.template and topology-group.data.template

  Original versions of configuration files, ASL rule sets, scripts, and template files are installed to the BASEDIR/smarts/conf, BASEDIR/smarts/rules, and BASEDIR/smarts/script directories. When you need to alter a file in one of these directories, make those changes to a local copy of the file. By default, local copies of user-modifiable files should reside in BASEDIR/smarts/local or one of its subdirectories. SDK software is designed to first search for user-modifiable files in BASEDIR/smarts/local or one of its subdirectories. If a modified version of a file is not found in the local area, SDK software then searches appropriate non-local directories.

  To edit a file, invoke the sm_edit utility from the BASEDIR/smarts/bin directory. When used, sm_edit automatically opens a local copy of the specified file and saves the modified file to its appropriate location. Modifying SDK files provides further information about how to use the sm_edit utility.

  You should:

- *Only* modify local copies of user-modifiable files.
- *Always* retain the original, unedited, version of these files.

  Subdirectories for user-modifiable files identifies the default subdirectories that include user-modifiable files found under BASEDIR/smarts/local. The subdirectories vary by product suite.

Table 3-2. Subdirectories for user-modifiable files

Installation path	Directory	Description
BASEDIR/smarts/local	/actions	Tool scripts
	/conf	Configuration files
	/data	Import files
	/rules	ASL rules files

Note The BASEDIR/smarts/local directory has been reserved for directories with local copies of user-modifiable files as well as writable files such as logs and repository files. It typically contains directories listed in Subdirectories for user-modifiable files. During installation, SDK software may create empty subdirectories under the BASEDIR/smarts/local directory. These directories are reserved for use for future service packs and Rolling Patches. The exception is the BASEDIR/smarts/local/bin directory that contains files necessary for the successful installation of SDK software.

In addition, the BASEDIR/smarts/local/conf directory contains the runcmd_env.sh file, which is created during installation. The runcmd_env.sh is a file for setting environment variables, including the default location for the Broker. You can edit this file to change the default location of the Broker or to define any SDK-wide environment variable. *"Methods for setting environment variables for ITOps software" on page 138* provides information about setting environment variables.

The product-specific naming convention used for subdirectories under BASEDIR/smarts/local/conf and BASEDIR/smarts/local/rules (as well as their non-local counterparts) typically reflects the SDK product or module within a product suite that uses the files.

You can alter the location for rules files, as described in Controlling the location for rule set files.

Note If VMware installs a site-specific patch at your location, then additional directories may be included under BASEDIR/smarts/local. Never alter files in any of these additional subdirectories.

Modifying SDK files

After you install an SDK application, you may need to modify files as part of the configuration process. For example, if you want to secure access to SDK software, you need to change the password for the Admin account in the serverConnect.conf and clientConnect.conf files.

To modify SDK files, use sm_edit, an SDK utility that is installed with SDK software.

Note You must use the sm_edit utility that is included with the installation of a product suite to modify files applicable to that suite. Do not, for example, edit IP files with the sm_edit that is installed with SAM.

When invoked, sm_edit opens the specified file in a text editor. This utility ensures that modified files are always saved to the appropriate local area and that non-local copies of all files remain unchanged. If an appropriate subdirectory does not exist for the file you are modifying, sm_edit creates the appropriate subdirectory before saving the modified file to that location. For files with header information set for encryption, sm_edit encrypts certain fields in the file. In addition, sm_edit preserves the file permissions of modified files, which helps ensure that important configuration files are not altered by unauthorized users.

To use sm_edit from the command line, specify the filename and include the subdirectory under /local where the file resides. For example, to edit the ics.conf file, type the following command by using administrative privileges:

```
BASEDIR/smarts/bin> sm_edit conf/ics/ics.conf
```

In this example, sm_edit searches in the BASEDIR/smarts/local/conf/ics directory for the ics.conf file. If it finds the ics.conf file, it opens the file in a text editor. If sm_edit does not find the ics.conf file in the BASEDIR/smarts/local/conf/ics directory, it creates a local copy of the ics.conf file and writes it to the BASEDIR/smarts/local/conf/ics directory.

If the appropriate subdirectory does not exist in the local area for the file you are modifying, sm_edit creates that subdirectory and saves the modified file there. This ensures that the original version of the file remains unchanged.

Specifying the text editor used by the sm_edit utility

The sm_edit utility is not a text editor. Instead, it uses the text editor specified by one of the following means:

- SM_EDITOR environment variable, as described in *“Environment variables for users” on page 140*.
- VISUAL environment variable
- EDITOR environment variable
- If the SM_EDITOR, VISUAL or EDITOR environment variables are not defined, sm_edit uses the following:
  - Edit program on UNIX

    You can also specify the editor that sm_edit should use by providing the --**editor** argument when invoking the utility.

How SDK software locates user-modifiable files

It is important to understand how SDK software locates user-modifiable files at runtime. In general, when looking for a user-modifiable file, SDK software first searches in BASEDIR/smarts/local or one of its subdirectories. If the file is not found, SDK software then proceeds to look for the file in directories that are not intended to contain modified files. For example, when SDK software searches for a configuration file, it first looks in BASEDIR/smarts/local/conf. If the file is not found, then SDK software proceeds to look in BASEDIR/smarts/conf.

Note BASEDIR/smarts can refer to different product suite directories. Therefore, the configuration file in this example could be a configuration file for any product suite such as the IP Manager or Service Assurance Manager.

Controlling the location for rule set files

To locate your ASL rule sets in an area other than the default location (BASEDIR/smarts/local/rules), set the SM_RULESET_PATH environment variable. When you set SM_RULESET_PATH, SDK software first searches for rule set files in that path location or in one of its subdirectories. If the file is not found, SDK software then proceeds to look in the default location for modifiable rule set files.

To set SM_RULESET_PATH, add it to the runcmd_env.sh file.

Note You normally only define the SM_RULESET_PATH environment variable if you are writing custom ASL rule sets to work with your SDK software.

Chapter 8 Environment Variables Used by VMware Software provides further information about SM_RULESET_PATH.

Where SDK software writes output files

The BASEDIR/smarts/local directory is also the default location for files written by SDK software. Writable files include logs, saved consoles, and repository files. If desired, you can alter the location for log files, as described in Controlling where SDK software writes log files. Default subdirectories for ITOps writable files defines the default subdirectories found under BASEDIR/smarts/local that contain writable files.

Table 3-3. Default subdirectories for SDK writable files

Installation path	Directory	Description
BASEDIR/smarts/local	/consoles	SDK consoles that have been saved.
	/logs	Manager and adapter log files that SDK software has written. Also where archived notifications are saved. Note that installation and uninstallation logs are written to the BASEDIR/smarts/setup/logs directory.
	/repos	Repository files that SDK software has written.

For Service Assurance Manager, the BASEDIR/smarts/local/logs directory includes a <manager_name>.audit file. This file contains audit entries that are added to all notifications in Service Assurance Manager. The entries, which are in time order, include the following information: date/time, notification name, source, user, and action.

The directory also includes a <manager_name>-statistics.log. This file contains entries for every invocation of the following operations:

```
dmctl -s <manager_name> invoke ICS_ActionManager::ICS-ActionManager dumpStatistics <filename>

dmctl -s <manager_name> invoke ICS_AutoActionManager::ICS-AutoActionManager dumpStatistics <filename>
```

The *<filename>* is optional; otherwise, it is *<manager_name>*-statistics.log.

The information in this file enables network personnel to monitor the execution of custom actions and the performance of administrator-defined escalations.

Controlling where SDK software writes output files

SDK software defines the location of writable files by using the SM_WRITEABLE environment variable. By default, SM_WRITEABLE is defined as BASEDIR/smarts/local.

This means the following:

- Saved consoles are written to BASEDIR/smarts/local/consoles.
- Log files are written to BASEDIR/smarts/local/logs.
- Repository files are written to BASEDIR/smarts/local/repos.

  If you want all SDK writable files to be written elsewhere, set the path by using the SM_WRITEABLE environment variable.

  To set SM_WRITEABLE, add the variable to the runcmd_env.sh file.

Controlling where SDK software writes log files

If you want SDK software to write log files to an area other than the default location, set the path by using the SM_LOGFILES environment variable.

To set SM_LOGFILES, add the variable to the runcmd_env.sh.

SM_LOGFILES takes precedence over SM_WRITEABLE, as described in SM_LOGFILES.

sm_logerror

On Solaris and Linux systems, every SDK process creates a child process, sm_logerror, at startup. This process has one purpose: to print a stack trace of its parent (by invoking the pstack program) should the parent request it (which usually happens only when the parent encounters a fatal error).

Every SDK process also starts a child running an external authenticator, sm_authority (in the default configuration). This program either gets client credentials to send to a Manager, or checks credentials received by a Manager. Since the authenticator itself is an SDK program, it starts its own sm_logerror. Thus, if you follow the tree of processes under an sm_server, you might find:

```
sm_server
 sm_logerror
 sm_authority
  sm_logerror
```

Both sm_logerror and sm_authority spend their time reading a pipe that connects them to the process that created them. However, this process is not necessarily their parent. When the Manager starts as a service:

- The Domain Manager runs as a daemon after starting sm_logerror and sm_authority.
- The original sm_server process is replaced by a different one.
- The child processes, sm_logerror and sm_authority, become orphans and are then inherited by the init process, which is always process 1.

  The new sm_server process retains its connection to the pipes that connected the original to the sm_logerror and sm_authority processes, and the processes work as intended

  These processes must not be "killed". Killing sm_logerror will make it impossible to print a stack trace, and make debugging more difficult. Killing sm_authority will cause connection attempts to fail when credentials cannot be obtained or checked.

Using the smgetinfo utility to save modifications

Each SDK product suite includes an smgetinfo utility. The utility enables you to maintain a backup copy of all essential files and customizations from your SDK installation. It also enables you to collect data about your current installation so that you can send the data to VMware Global Services for support purposes. The smgetinfo utility is located in the BASEDIR/smarts/script directory.

When run, the smgetinfo utility stores the following SDK directories in compressed files:

- /conf
- /local
- /rules
- /setup

  It supports the following formats:

- For UNIX: .tar.gz
- For UNIX: .tar.Z file
- Versions and locations of all SDK installed products
- Version of SDK

- Version of dmctl
- Versions of all executables in BASEDIR/smarts/bin/system
- Libraries in the BASEDIR/smarts/lib directories
- Information about the host, operating system, and CPU

  To run the smgetinfo utility, change to the BASEDIR/smarts/script directory and execute the following command with administrative privileges:

  On UNIX:

  ```
  smgetinfo
  ```

  or

  ```
  sh smgetinfo
  ```

Controlling the Startup of VMware Smart Assurance Software

4

This chapter includes the following topics:

- Root access requirement on Unix and Linux
- Overview of services
- About the sm_service utility
- sm_serviced and ic-serviced
- The sm_service command line
- Installing programs as services with sm_service
- Displaying installed services and their status
- Starting and stopping services with sm_service
- Modifying service parameters with sm_service
- Removing services with sm_service
- Default parameters for VMware services
- Starting a Domain Manager
- Starting other DMT processes
- Running as non root on UNIX and Linux systems

Root access requirement on Unix and Linux

Super-user access is required on Unix and Linux systems for running the following:

- sm_server - Domain Manager, handling ICMP packets, SNMP trap receiving on default port 162
- brstart - Broker on default port 426
- sm_trapd - SNMP trap receiving on default port 162

  A Domain Manager must bind to multiple low numbered (less than 1024) ports for listening for ICMP traffic. In addition, the default port for starting a Broker (426) and SNMP trap processing (162) are also low-numbered ports. Access to low-numbered ports requires root access on Unix and Linux systems.

You can also start a Domain Manager with root access and then open additional ports as a non-root user, if this is required. *Running as non root on UNIX and Linux systems on page 82* provides additional information.

Overview of services

SDK programs can be installed as services. A service is a program that, once started, is intended to run continuously. A service is administered by the sm_service utility. SDK programs installed as services start automatically upon system reboot; those not installed as services (manual processes or disabled processes) require that you issue commands to start and stop them as necessary.

VMware recommends that you install SDK programs as services. You are prompted to make this choice during the software installation. When you choose to install a program as a service, the installation process automatically sets up the program accordingly.

The following are examples of VMware programs that should be installed as services:

- Broker
- Service Assurance Global Manager
- Service Assurance Adapter Platform (including the SNMP Trap Adapter and Syslog Adapter)
- IP Availability Manager and IP Performance Manager

  Although services start automatically upon system reboot, there will be occasion for you to manually start (and stop) a program that was installed as a service. *Starting and stopping services with sm_service on page 74* describes starting and stopping services.

  If you choose not to install a program as a service, the program is installed as a manual process. If you install a program as a manual process, you can later change it to run as a service, described in *Installing programs as services with sm_service on page 71*.

  In addition, you can modify a service™s settings, as described in *Modifying service parameters with sm_service on page 76*.

Note The sm_service utility cannot be used with the Failover System. The VMware Smart Assurance Failover System User Guide provides specific information about starting, stopping, and configuring the Failover System scripts.

About the sm_service utility

The sm_service is a cross-platform utility that sets up the environment for standard SDK applications (for example, the Broker, Domain Managers, and adapters), and that installs those applications as services.

Additionally, the sm_service utility can start non-SDK applications.

The implementation of the sm_service utility on:

- UNIX includes two separate programs: sm_serviced, a long-running, system-level program, and sm_service, a command-line tool that communicates user requests to sm_serviced.

  The sm_service utility is installed with the SDK software (one per system), and is used from the BASEDIR/smarts/bin directory. Select the BASEDIR/smarts with the latest version of software. The utility can only be used to control services on the local machine.

sm_serviced and ic-serviced

UNIX includes two separate programs: sm_serviced, a long-running, system-level program, and sm_service, a command-line tool that communicates user requests to sm_serviced.

The sm_serviced, the component of the sm_service utility that manages programs installed as services (either at installation, or by way of the sm_service command line), is installed with the SDK software on UNIX systems during the installation process, and automatically starts its operations.

Thereafter, a script, ic-serviced, can be used to start and stop sm_serviced. The script is stored in a system-specific location: /etc/init.d on Solaris and Linux. The control of services can be affected through sm_service. *The sm_service command line on page 68* provides additional information.

The ic-serviced script includes several variables. Default settings are assigned to the variables during the installation process. If necessary, the value for the SM_SERVICE_STARTDIR variable can be edited. The default setting for this variable is / and specifies the directory into which core files are written. If it is changed, the setting must point to a directory on a file system with enough free space to hold a core file of the system™s largest server.

Note Contact VMware Global Services for information about editing the remaining variables at the beginning of the ic-serviced script.

The ic-serviced script enables you to start and stop sm_serviced on UNIX systems and to check the status of sm_serviced.

If sm_serviced is not already running, issue the following command with administrative privileges to start it:

```
<system-dependent path>/ic-serviced start
```

To stop sm_serviced and all of the services managed by it, issue the following command with administrative privileges:

```
<system-dependent path>/ic-serviced stop
```

To check that sm_serviced is running and responding, issue the following command with administrative privileges:

```
<system-dependent path>/ic-serviced status
```

The sm_service command line

The sm_service command line is the tool through which you communicate user requests to sm_serviced. The basic syntax for the command line is:

```
sm_service <action> <options> <arguments>
```

Standard options

Standard options for sm_service lists the standard options that can be used with the sm_service utility.

Table 4-1. Standard options for sm_service

Options	Description
--help	Print the help text and exit.
--version	Print the program version and exit.
--logname=<*name*>	Use <*name*> to identify sender in the system log. Default: The program's name.
--loglevel=<*level*>	Minimum system logging level. The level can be: None, Emergency, Alert, Fatal, Critical, Error, Warning, Notice, Informational, Debug. Note that Fatal is equivalent to Emergency, Alert, or Critical. Default: Error.
--errlevel=<*level*>	Minimum error printing level. The level can be: None, Emergency, Alert, Fatal, Critical, Error, Warning, Notice, Informational, Debug. Note that Fatal is equivalent to Emergency, Alert, or Critical. Default: Warning.
--tracelevel=<*level*>	Minimum stack trace level. The level can be: None, Emergency, Alert, Fatal, Critical, Error, Warning, Notice, Informational, Debug. Note that Fatal is equivalent to Emergency, Alert, or Critical. Default: Fatal.
--output[=<*file*>]	Redirect output (stdout and stderr). The filename is <*file*>, or the --logname value if <*file*> is omitted. Log files are always placed in $SM_LOGFILES or $SM_WRITEABLE/logs.

Table 4-1. Standard options for sm_service (Continued)

Options	Description
--facility=<*facility*>	Instructs the VMware program to log any syslog messages with the specified syslog facility (UNIX/Linux only). The default is Daemon. The <*facility*> setting can be one of the following: ■ Kern ■ User ■ Mail ■ Daemon ■ Auth ■ Lpr ■ News ■ Uucp ■ Cron ■ Local0 ■ Local1 ■ Local2 ■ Local3 ■ Local4 ■ Local5 ■ Local6 ■ Local7
--	Stop scanning for options.

The following example highlights the use of a standard option. You must type it with administrative privileges.

```
    BASEDIR/smarts
/bin/sm_service --help
```

sm_service actions

sm_service actions lists the actions that can be performed with the sm_service utility.

Table 4-2. sm_service actions

Actions	Description
install	Installs a program as a service. It can also be used to modify the parameters of a service.
show	Displays the status of the installed services. It can also display the command lines of specified services so that they can be replicated on other systems.
start	Starts one or more installed services.
stop	Stops one or more installed services.
isstopped	Exits with a non-zero status if the requested set of services is in a state of not running.
remove	Removes one or more installed services.

The following example highlights an action with several install options. You must type it with administrative privileges.

```
t
            BASEDIR/smarts
         /bin/sm_service install --startmode=runonce --name=ic-broker
            BASEDIR/smarts
         /bin/brstart
 --port=1234 s
```

Installing programs as services with sm_service

If, at installation, you installed a program to run as a service, the service will automatically start up when your system reboots. If you did not install a program as a service, you can use sm_service to install it as a service at any time.

The install action enables you to install a program as a service. After you define the name, description, startmode, and path of the program, sm_service stores the information in an sm_service database located in the /var/smarts directory (on UNIX). The sm_service then uses the information to start and run the program as a service. You must have administrative privileges to type this command.

The syntax for the sm_service install action is:

```
tsm_service install [<install options>] <name> path/to/exe
[<args>...] s
```

In addition to the common install options, such as name, description, and startmode, the command line must include an absolute path to the program you want to the program that you want to install as a service. Arguments that apply to the program, that you are installing as a service, can follow the path.

For example:

UNIX

```
t
            BASEDIR/smarts
         /bin/sm_service install --startmode=runonce --name=ic-broker
            BASEDIR/smarts
         /bin/brstart --port=1234 s
```

Install options

Install options for sm_service lists the install options that can be used with the sm_service utility.

Table 4-3. Install options for sm_service

Options	Description
--force	Overwrite an existing service with the same name. The option is used to update or modify the parameters of an existing service.
"--description=<*DESC*>"	A short description of the service. Enclose the option with double quotation marks.
--startmode=<*mode*>	The service start policy. Where <*mode*> is one of the following: ■ runonce (start automatically when sm_serviced starts) ■ automatic (starts automatically when sm_serviced does not detect that it is running) ■ manual (requires an explicit start request) ■ disable (cannot be started). Default: runonce
--env=<NAME>=<*VALUE*>	A <*NAME*>=<*VALUE*> pair which will be placed in the process environment of the launched service. As many --env pairs as necessary can be specified. <*VALUE*> may contain one or more values, separated by commas. The syntax <*NAME*>= (with no specified value) has the effect of unsetting <*NAME*> in the environment of the launched program. The --env arguments are applied left to right as they appear on the command line, and this ordering is preserved in the database.

Examples of the sm_service install action

The following provides examples of sm_service install action command lines:

UNIX

```
t
            BASEDIR/smarts
         /bin/sm_service install --startmode=runonce
--name=ic-broker BASEDIR/smarts/bin/brstart --port=1234 s
t
            BASEDIR/smarts
         /bin/sm_service install --startmode=runonce
--env=SM_BROKER=localhost:1234 trapadapter
BASEDIR/smarts/bin/sm_trapd s
```

Displaying installed services and their status

On UNIX systems, the sm_service show action displays previously installed services and their status.

The syntax for the action is as follows. You must type this command with administrative privileges:

```
sm_service show <name> [<name> ...]
```

The show action has one option:

```
--cmdline
```

When the action is used with its **--**cmdline option, it displays the sm_service command line that installed the program as a service. The command line can be reproduced and used for installing the requested service(s) on another system.

In the absence of the **--**cmdlineoption, the show action displays the status of the named service (or all services, when <name> is not given), including:

- Whether the service is running
- The name of the service
- A brief description of the service

Examples of the sm_service show action

The following provides an example of the sm_service show action.

For the status of a service, type the following with administrative privileges:

```
            BASEDIR/smarts
         /bin/sm_service show ic-broker
```

Output:

```
RUNNING ic-broker
```

For the command line of a service, type the following with administrative privileges:

```
            BASEDIR/smarts
         /bin/sm_service show --cmdline ic-broker
```

Output:

```
sm_service install --force --unmanaged --startmode=runonce\
 '--name=ic-broker' \
 '--description=VMware Smart Assurance
        Broker' \
 '--env=SM_CLIENTCONNECT=brokerConnect.conf' \
 '
            BASEDIR/smarts
        /bin/brstart' \
 '--port=426' \
```

```
'--restore=
          BASEDIR/smarts
       /local/repos/broker/broker.rps' \
'--output'
```

Starting and stopping services with sm_service

The sm_service utility can be used to start and stop programs when necessary. However, the programs, which can be either manual processes or services, must be installed with sm_service before you can use this action.

To start a program, type the following command with administrative privileges:

```
sm_service start [options]
```

To stop a program, type the following command with administrative privileges:

```
sm_service stop [options]
```

Start options

Start options for sm_service lists the options for the start action.

Table 4-4. Start options for sm_service

Options	Description
<name> [*<name>*...]	Start specified processes
--all	Start all automatic or runonce services that are not running. (UNIX only).
--pattern=	Start all processes with absolute paths that match the wildcard pattern.

Stop options

Stop options for sm_service lists the options for the stop action.

Table 4-5. Stop options for sm_service

Options	Description
<name> [*<name>*...]	Stop specified processes
--all	Attempt to stop all running services (UNIX only).
--pattern=	Stop all processes with absolute paths that match the wildcard pattern.
--force	Do not consider it an error if the service is not running.

Examples of the sm_service start and stop actions

The following provides examples of the sm_service start and stop actions.

For the start action, type the following command with administrative privileges:

```
   BASEDIR/smarts
/bin/sm_service start ic-broker
```

For the stop action, type the following command with administrative privileges:

```
   BASEDIR/smarts
/bin/sm_service stop ic-broker
```

Checking whether services are stopped

The isstopped action for sm_service exits with a non-zero status if any of the requested set of services are not running.

Isstopped options for sm_service lists the options for the isstopped action.

Table 4-6. Isstopped options for sm_service

Options	Description
<name> [*<name>*...]	Return status for specified processes
--all	Return the status of all processes.
--pattern=	Return the status of all processes with absolute paths that match the wildcard pattern.

Examples of the sm_service isstopped action

The following provides examples of the sm_service isstopped actions on UNIX. These examples also show the use of wildcards.

The isstopped action verifies whether all of the services started from the BASEDIR/smarts directory are stopped:

```
sm_service isstopped --pattern '
   BASEDIR/smarts
/*'
```

This example checks if all Managers are stopped:

```
sm_service isstopped --pattern '*sm_server*'
```

Modifying service parameters with sm_service

The parameters for a service can be modified with the sm_service install action. *Installing programs as services with sm_service on page 71* provides additional information. The **--**force option, which overwrites the existing parameters, must be used with the install action when you modify a service.

The syntax for the sm_service install action (to modify service parameters) is:

```
sm_service install --force
```

You must type this command with administrative privileges.

To modify service parameters:

1 Stop the existing service with the **sm_service stop** action.

2 Use the **sm_service show** action with the **--**cmdline option to display the command line that was used to install the existing service.

3 Use the **sm_service install** action with the **--**force option, copy the displayed command line from step 2 after the **--**force option, and modify the parameters that need to be changed.

4 Start the modified service.

Removing services with sm_service

One or more existing services can be removed from the system, when necessary, with the sm_service remove action.

The syntax for the sm_service remove action is:

```
sm_service remove <name> [<name>...]
```

You must have administrative privileges to type this command.

Example of the sm_service remove action

The following provides an example of the sm_service remove action.

```
            BASEDIR/smarts
         /bin/sm_service remove trapadapter
```

You must have administrative privileges to type this command.

Default parameters for VMware services

During installation, SDK products can be installed as services. When installed as services, default values are specified for the parameters that are associated with the services.

This section lists the default parameters associated with SDK products, and briefly describes them. The installation guide that accompanied your product suite provides examples of using the sm_service install command.

If you need to modify the default parameters of a currently installed service, use the steps found in *Modifying service parameters with sm_service on page 76*.

Common sm_server options

Several sm_server options are common to all Managers that are installed as services (the IP Availability Manager, for example, and the Service Assurance Manager). Common sm_server options lists these sm_server options.

Exceptions to these parameters are noted and described with the default parameters for the individual SDK products.

Table 4-7. Common sm_server options

Options	Description
--name=<*name*>	Specifies the name of the Manager (Manager or Global Manager).
--config=<*cfg*>	Specifies the sm_server configuration file to use. Configuration files are loaded from the directories BASEDIR/smarts/local/conf/<*cfg*> and BASEDIR/smarts/conf/<*cfg*>.
--port=<*port*>	Specifies the alternate Manager port.
--bootstrap=<*file*>	Specifies the alternate bootstrap configuration file. The file name is relative to the configuration directory (as set by the --configoption). The default is bootstrap.conf.
--ignore-restore-errors	Indicates ignore errors encountered while restoring the saved topology.
--norestore	Indicates do not attempt to restore any saved objects.
--subscribe=<*sub*>	Start a subscription adapter that automatically subscribes to the specified notifications.
--broker=<*location*>	Specifies the alternate Broker location as host:port. Also -b <*location*>.
--output[=<*file*>]	Redirects the output (stdout and stderr). The file name is <*file*>, or the **--l**ogname value if <*file*> is omitted. Log files are always placed in BASEDIR/smarts/local/logs.

Table 4-7. Common sm_server options (Continued)

Options	Description
--daemon	Run the SDK program as a daemon (UNIX only). Messages generated from internal logging calls are directed to the log file. Log files are written to the BASEDIR/smarts/logs directory.
--facility=<*facility*>	Instructs the VMware program to log any syslog messages with the specified syslog facility (UNIX/Linux only). The default is Daemon. The <*facility*> setting can be one of the following: ■ Kern ■ User ■ Mail ■ Daemon ■ Auth ■ Lpr ■ News ■ Uucp ■ Cron ■ Local0 ■ Local1 ■ Local2 ■ Local3 ■ Local4 ■ Local5 ■ Local6 ■ Local7

To display a complete list of the sm_server options on your system, type the following command with administrative privileges:

```
BASEDIR/smarts
/bin/sm_server --help
```

Common sm_adapter options

Several sm_adapter parameters are common to all adapters that are installed as services (the Syslog Adapter, for example, and the SNMP Trap Adapter). Common sm_adapter options lists these sm_adapter parameters or options.

Exceptions to these parameters are noted and described with the default parameters for the individual SDK products.

Table 4-8. Common sm_adapter options

Options	Description
--name=<*name*>	Specifies the name of the adapter.
--model=<*name*>	Specifies the name of model library to load.

Table 4-8. Common sm_adapter options (Continued)

Options	Description
--tail=*<path>*	Specifies read input by tailing a file.
--rserver=*<name>*	Specifies auto-reconnect parser to Manager.
--server=*<name>*	Specifies connect parser to Manager.
--broker=*<location>*	Specifies the alternate Broker location as host:port. Also -b *<location>*.
--output=*<file>*	Directs the output (stdout and stderr). The file name is *<file>*, or the --logname value if *<file>* is omitted. Log files are always placed in BASEDIR/smarts/local/logs.
--daemon	Run the SDK program as a daemon (UNIX only). Messages generated from internal logging calls are directed to the log file. Log files are written to the BASEDIR/smarts/logs directory.
--facility=*<facility>*	Instructs the SDK program to log any syslog messages with the specified syslog facility (UNIX/Linux only). The default is Daemon. The *<facility>* setting can be one of the following: ■ Kern ■ User ■ Mail ■ Daemon ■ Auth ■ Lpr ■ News ■ Uucp ■ Cron ■ Local0 ■ Local1 ■ Local2 ■ Local3 ■ Local4 ■ Local5 ■ Local6 ■ Local7

To display a complete list of the sm_adapter options on your system, type the following command with administrative privileges:

```
BASEDIR/smarts
/bin/sm_adapter --help
```

Common sm_sdi options

Several sm_sdi options are common to the components of the SQL Data Interface Adapter that are installed as services. Common sm_sdi options lists these sm_sdi options.

Exceptions to these parameters are noted and described with the default parameters for the individual SDK products.

Table 4-9. Common sm_sdi options

Options	Description
--broker=<*location*>	The alternate Broker location as host:port. Also -b <*location*>.
--name=<*name*>	The name of the SQL Data Interface Adapter component. Also -n <*name*>.
--config=<*cfg*>	Specifies the directory of the sm_sdi configuration files: sdi_ics.conf, sdi_sql.conf and sdi_odbc.conf. Configuration files are loaded from the directories BASEDIR/smarts/local/conf/<*cfg*> and BASEDIR/smarts/conf/<cfg>.
--bootstrap=<*file*>	Specifies the alternate bootstrap configuration file. The filename is relative to the configuration directory (as set by the --configoption). The default is bootstrap.conf.
--nonpriv	Allow sm_sdi to run under a nonprivileged user ID. This option must be specified in order for a nonprivileged user to start sm_sdi.

To display a complete list of the sm_sdi options on your system, type the following command with administrative privileges:

```
BASEDIR/smarts
/bin/sm_sdi --help
```

Starting a Domain Manager

The sm_server command can be used to start a Domain Manager. The syntax of the command is as follows:

```
sm_server [<options>...]
```

Some of the common sm_server options are described in Table 7 on page 60.

Table 4-10. Common sm_server options

Options	Description
--name=<*name*>	Specifies the name of the Manager.
--config=<*cfg*>	Specifies the sm_server configuration file to use. Configuration files are loaded from the directories BASEDIR/smarts/local/conf/<*cfg*> and BASEDIR/smarts/conf/<*cfg*>.
--port=<*port*>	Specifies the alternate Manager port.
--bootstrap=<*file*>	Specifies the alternate bootstrap configuration file. The file name is relative to the configuration directory (as set by the --config option). The default is bootstrap.conf.
--ignore-restore-errors	Indicates ignore errors encountered while restoring the saved topology.
--norestore	Indicates do not attempt to restore any saved objects.
--subscribe=<*sub*>	Specifies start a subscription adapter that automatically subscribes to the specified notifications.
--broker=<*location*>	Specifies the alternate Broker location as host:port. Also -b <*location*>.

Table 4-10. Common sm_server options (Continued)

Options	Description
--output[=<*file*>]	Messages generated from internal logging calls are directed to the log file. The file name is <*file*>, or the --logname value if <*file*> is omitted. Log files are always placed in BASEDIR/smarts/local/logs.
--daemon	Run the SDK program as a daemon (UNIX only). Messages generated from internal logging calls are directed to the log file. Log files are written to the BASEDIR/smarts/logs directory.
--facility=<*facility*>	Instructs the DMT program to log any syslog messages with the specified syslog facility (UNIX/Linux only). The default is Daemon. The <*facility*> setting can be one of the following: ■ Kern ■ User ■ Mail ■ Daemon ■ Auth ■ Lpr ■ News ■ Uucp ■ Cron ■ Local0 ■ Local1 ■ Local2 ■ Local3 ■ Local4 ■ Local5 ■ Local6 ■ Local7

To display a complete list of the sm_server options on your system, type the following command with administrative privileges:

BASEDIR/smarts /bin/sm_server --help

Starting other DMT processes

The SDK provides additional commands for starting Brokers and adapters.

Broker

The brstart command is used to start a Broker. The syntax of the brstart command is as follows:

```
brstart [<options>...]
```

brstart options describes the options available for the brstart command.

Table 4-11. brstart options

Options	Description
--backup=<*file*>	Backup file for Broker state; default BASEDIR/smarts/repos/broker/broker.rps.
--restore=<*file*>	Restore Broker state from a backup file. Also -R <*file*>.
--port=<*port*>	Alternate Broker port; default 426. Also -p <*port*>.

Adapter

The sm_adapter command can be used to start an adapter. Common sm_adapter options describes the common sm_adapter options.

Table 4-12. Common sm_adapter options

Options	Description
--name=<*name*>	Specifies the name of the adapter.
--model=<*name*>	Specifies the name of model library to load.
--tail=<*path*>	Specifies read input by tailing a file.
--rserver=<*name*>	Specifies auto-reconnect parser to Manager.
--server=<*name*>	Specifies connect parser to Manager.
--broker=<*location*>	Specifies the alternate Broker location as host:port. Also -b <*location*>.
--output=<*file*>	Messages generated from internal logging calls are directed to the log file. The file name is <*file*>, or the --logname value if <*file*> is omitted. Log files are always placed in BASEDIR/smarts/local/logs.

Table 4-12. Common sm_adapter options (Continued)

Options	Description
--daemon	Run the SDK program as a daemon (UNIX only). Messages generated from internal logging calls are directed to the log file. Log files are written to the BASEDIR/smarts/logs directory.
--facility=<*facility*>	Instructs the DMT program to log any syslog messages with the specified syslog facility (UNIX/Linux only). The default is Daemon. The <*facility*> setting can be one of the following: ■ Kern ■ User ■ Mail ■ Daemon ■ Auth ■ Lpr ■ News ■ Uucp ■ Cron ■ Local0 ■ Local1 ■ Local2 ■ Local3 ■ Local4 ■ Local5 ■ Local6 ■ Local7

To display a complete list of the sm_adapter options on your system, type the following command with administrative privileges:

```
BASEDIR/smarts
/bin/sm_adapter --help
```

Running as non root on UNIX and Linux systems

Additional options for running as non rooton UNIX and Linux systems describes two options that are used in conjunction with each other to run as non root. These two options are only available on UNIX and Linux systems. These options are available for the sm_server, brstart, and sm_adapter commands.

Table 4-13. Additional options for running as non rooton UNIX and Linux systems

Options	Description
--privopen=<*arg*>[,<*arg*>]	**Note** This option is only used in conjuction with the --run-as-user option to run as non root. Opens privileged sockets. The <*arg*> parameter has the following syntax: <*type*>[:<*family*>]:<*port*>[,<*count*>] where: ■ <*type*> is one of the following: TCP (for a TCP connection) UDP (for a datagram) IP (for a raw socket) ■ <*port*> is one of the following: The required privileged port for a TCP socket The required privleged port for a UDP socket The protocol for IP ■ <*family*> is one of the following: :v4 (IPv4 address family) :v6 (IPv6 address family) If <*family*> is not specified, the address family defaults to IPv4. ■ <*count*> is the number of sockets of the type, family and port. The default count is 1.
--run-as-user=<*username*>	**Note** This option is only used in conjuction with the --privopen option to run as non root. Specifies a valid user name.

The --privopen and --run-as-user options are used in conjunction to run a Domain Manager with non root privileges. The Domain Manager can startup as root, open privileged ports, and change to a non root user.

You can open multiple ports by repeating the <*arg*> parameter. Each instance is separated by a comma (,). For example:

```
--privopen=UDP:v4:161,2
--privopen=IP:1,4
--privopen=IP:v6:1
--run-as-user=testuser1
```

In this example:

- The first privopen line opens two UDP IPv4 sockets on port 161.
- The second privopen line opens four raw IPv4 sockets for ICMP.
- The third privopen line opens one raw IPv6: socket for ICMP.

  After the sockets are opened, the process will change to run as user, "testuser1".

Stopping DMT programs

This section describes how to stop active DMT programs.

Stopping active DMT programs (UNIX)

1 Determine if any Domain Managers are running by using the brcontrol utility from *BASEDIR/smarts* /bin:

```
# ./brcontrol -b <host>:<port>
```

Table 4-14. Next steps for detecting and stopping programs

The Broker is on the	And brcontrol displays.	Do this
Same host	"Error attaching to broker" message	The Broker is not running. Go to step 3 on page 65 .
	List of servers registered with the Broker	■ The Broker did not shut down. ■ If any servers are still running on the host, these did not shut down. Go to step 2 on page 65
Different host	List of servers registered with the Broker	■ If any servers are still running on the host where you will install the SDK, these did not shut down. Go to step 2 on page 65 .

1 Stop any Domain Manager that is still running. If the Broker is on the host, stop the local Broker:

- To stop any manager that is still running, use the dmquit utility from *BASEDIR/smarts* /bin:

```
# ./dmquit --server=<
                   servername
                   >

# ./brquit --broker=localhost:<port>
```

2 Determine if any other DMT processes are still running and shut the processes down:

- Detect the processes using the following command:

```
# ps -ef | grep sm_
# kill <pid>
```

Securing Access to VMware Software

5

This chapter includes the following topics:

- SDK security features
- How SDK security works
- Configuring authentication
- Controlling authentication
- Encryption
- Encrypted passwords
- Encrypted seed files
- Encrypted connections
- Configuring a secure Broker
- Additional safeguards
- Example security configurations

SDK security features

SDK software provides multiple levels of security which SDK administrators can use to secure their system and control access to it. The various levels of security can be implemented through:

- Authentication and user privileges

  Client-server connections can be controlled on both the client and server sides of the system. The system is secured by using authentication records and by assigning connection privileges on the server side. When a client initiates a connection to a server, the client must supply appropriate authentication to the server before the connection (as defined by the connection privileges) is permitted. *"Configuring authentication" on page 89* provides detailed information.

- Encrypted passwords

  Passwords can be encrypted in the serverConnect.conf, clientConnect.conf, and brokerConnect.conf files. Encryption is based on a secret phrase, common to all of the SDK applications that must interact, and is used to encrypt password fields in the authentication records. *"Encrypted passwords" on page 102* provides detailed information.

  Passwords can also be encrypted in seed files. *"Encrypted seed files" on page 104* provides detailed information.

- Encrypted connections

  SDK software can provide three levels of security above cleartext communication: Diffie Helman-Advanced Encryption Standard (DH-AES), encryption based on the site secret, and DH-AES used in conjunction with the site secret. The levels of security are configured by way of values for environment variables. *"Encrypted connections" on page 105* provides detailed information.

  The SDK security features are enabled with default settings when you install SDK software. The default security settings are thoroughly documented and permit access to the system. As a result, you should change the default usernames, passwords, and the secret phrase to enforce access control to SDK software. In addition, you should restrict access to the security configuration files, as described in Specifying alternate security configuration files.

  Note An SDK administrator can also place access restrictions on certain console operations by applying user profiles. This ability to restrict SDK users to certain operations is described in the VMware Smart Assurance Service Assurance Manager Configuration Guide.

How SDK security works

SDK security features apply to authentication and communication connections.

Authentication occurs whenever a client program initiates a connection to a server program. The client passes an SDK username and a password to the server. The server determines whether the client is allowed to connect and, if the connection is allowed, what privileges the client is granted.

Secure communication connections are implemented through the use of the Diffie Helman-Advanced Encryption Standard (DH-AES), encryption based on a site secret, or DH-AES used in conjunction with the site secret.

To properly configure the security system, you must understand how the security system works. The answers to the following questions will help:

- Which programs are servers and which are clients?
- How does a server authenticate a client?
- How does a client obtain a username and password to pass on to a server?

- How does encryption apply to authentication as well as communication connections?

Note For the UNIX account used to run SDK processes, if the system administrator sets explicit permissions for the file mode creation mask (the umask) for a parent process, SDK software respects those permissions. For processes started by the SDK sm_serviced program described in *sm_serviced and ic-serviced on page 67*, the umask permission is inherited.

Server and client programs for SDK software

Knowing which SDK programs function as servers and which function as clients can help you understand how to configure security. Simply stated, a client is any SDK program that initiates a connection to another SDK program. Programs can act both as a client and as a server.

ITOps server and client programs lists some server and client programs and shows how they can interact.

Table 5-1. SDK server and client programs

Server programs	Client programs
Broker	Service Assurance Failover System
	Command-line utilities such as dmctl and brcontrol
	Global Console users
	Global Manager
	Notification Adapters
	Service Assurance Adapter Platform
	Tools
	Report Manager
	SDK Beacon
	Application Connectivity Monitor
Broker	Command-line utilities such as dmctl and brcontrol
	Notification Adapters
	Tools
	Report Manager
	SDK Beacon
Global Manager	Service Assurance Failover System
	Broker
	Command-line utilities (dmctl)
	Global Console users
	Notification Adapters
	Tools
	Global Manager
	Report Manager

Table 5-1. SDK server and client programs (Continued)

Server programs	Client programs
Service Assurance Adapter Platform	Broker
	Command-line utilities (dmctl)
	Global Manager
	sm_ems
	SNMP Trap Adapter
	Syslog Adapter
IP Availability Manager	Global Console
IP Performance Manager	Global Console
IP Discovery Manager	Global Console
SDK Beacon	Global Console
Application Connectivity Monitor	Global Console

Note The Broker functions as both a server and a client. In addition, when Service Assurance Managers are deployed in a multi-tiered environment, a Service Assurance Manager may also function as both a server and a client.

Note that the server and client programs listed in ITOps server and client programs can be installed on the same host. For example, it is common for the Broker and a Service Assurance Manager to run on the same host.

Server authentication

When a client program initiates a connection to a server, it must provide an SDK username and password. A server authenticates a connection request by comparing the SDK username and password it receives from a client to the authentication records in its configuration file, serverConnect.conf. The server uses the first authentication record that matches the information sent by the client.

The authentication records are reread for each attempted connection so any changes to the file will take effect immediately.

The Global Console always displays a log-on dialog box where a user must type an SDK username and password. Other SDK applications can prompt for an SDK username and password or be configured to automatically send the username and password.

For example, when a Global Console connects to a Service Assurance Manager, it sends an SDK username and password to the Service Assurance Manager. The Service Assurance Manager compares the credentials to the authentication record listed in its serverConnect.conf file. If the first matching record allows the connection, the Service Assurance Manager accepts the connection and grants the user the privileges specified by the authentication record. If the username and password do not match an authentication record, the connection is refused.

Client authentication

Other than the Global Console, most Most clients, by default, automatically send authentication information to the server. This information is stored in a clientConnect.conf file. A client that uses automatic authentication reads the records in the order that they appear, selecting the first record whose login username matches the user that runs the client and whose target matches the name of the server that is being connected to. Once it finds a match, the client sends the SDK username and password to the target server as authentication credentials. If the authentication succeeds, the server tells the client its access privilege. If the authentication fails, the server refuses the connection and the failure is recorded in the server™s log file.

Client authentication files are reread for each attempted connection. As such, you can edit the configuration files at any time and the changes take effect immediately.

The Broker uses its own client connection file, brokerConnect.conf.

For example, when a Domain Manager (such as an VMware Smart Assurance IP Availability Manager) registers with a Broker, the Domain Manager sends an SDK username and password from its clientConnect.conf to the Broker. The Broker checks the username and password against the records in its serverConnect.conf. Based on the results, it will grant or deny a connection.

However, when the Broker checks whether a registered Domain Manager is alive (by pinging the manager), the Broker must authenticate with that Domain Manager. To do this, it finds an SDK username and password in its brokerConnect.conf file to send to the Domain Manager. The Domain Manager checks for the username and password in its serverConnect.conf and either grants or denies the right for the Broker to ping it.

Encryption

By default, encryption is enabled during the installation process. All SDK suites and applications that interact with each other must share a common secret phrase. The security configuration files and files with passwords to SNMP v3 devices all contain encrypted information.

The basis for encryption is a secret phrase that gets transformed into the imk.dat file. This file is the basis of encryption for SDK programs. The programs use the site secret to encrypt passwords in the configuration files and keywords in seed files, as well as to encrypt connections between SDK programs.

Configuring authentication

This section describes the components of authentication. It covers the content and syntax of the security files, the default authentication records provided in each file, and how to specify alternate security configuration files.

Syntax of the security files

This section describes the syntax of the serverConnect.conf, clientConnect.conf, and brokerConnect.conf files.

Each file consists of one or more authentication records, each of which contains four fields. Note the following:

- A line that starts with two forward slashes (//) or a pound sign (#) is considered a comment and ignored.
- The fields of each authentication record are separated by colons (:).
- Any white space before, after, or between fields, is ignored. If the field value contains a space, you need to add an escape character, the backslash (\), before the space. When a backslash is encountered, the following character loses any special significance and is used as is.

  Note The first line of each of the configuration files contains encryption information. Do not change this line unless you want to disable encryption. To comment out the line, add a second pound sign.

  The clientConnect.conf, brokerConnect.conf, and serverConnect.conf are read from top to bottom. For example, when a client selects a record for automatic authentication, it uses the first record with a matching target and SDK username. If that username is denied a connection by the server, the client does not try again. The ordering of authentication records is important because you can use wildcards for certain fields. In general, more specific authentication records should be listed first. Chapter 9 Wildcards Used by VMware Software provides additional information about wildcards.

Server authentication records

Server authentication involves setting up authentication records in the serverConnect.conf file. Each record contains information to control access to a particular server, an SDK username and password, and a connection privilege. To modify this file, use the sm_edit utility as described in Modifying SDK files.

Note Always use sm_edit utility to edit the serverConnect.conf. This is particularly important if the passwords are encrypted. When you use the sm_edit utility to edit the file, it saves changes to the file in BASEDIR/smarts/local/conf and encrypts the passwords; it does not modify the default, non-encrypted version of the file.

Description of serverConnect.conf

The serverConnect.conf file defines who can connect to which server and what privileges they are granted. By default, separate versions of the file reside in the BASEDIR/smarts/conf and the BASEDIR/smarts/local/conf directories on the system where the server is running. The first version does not contain encrypted passwords, so the default values are accessible by anyone who can read the file. The version in BASEDIR/smarts/local/conf contains encrypted passwords.

The format of a record in serverConnect.conf is:

```
<target>:<
            SDK

          username>:<password>:<privilege>
```

Field descriptions for serverConnect.conf describes the four fields of an authentication record in the serverConnect.conf file.

Table 5-2. Field descriptions for serverConnect.conf

Field	Definition	Value
<target>	Name of the server for which this connection is intended. A server will only read this line if its name matches the value of the target field.	Can be a matching pattern with wildcards or one of the following special values: ■ <BROKER> indicates that this record applies only to the Broker. ■ ~<BROKER> indicates this record applies to all servers except the Broker.
<SDK username>	SDK username for the client requesting a connection.	Can be a matching pattern with wildcards or the following special value: ■ <DEFAULT> is provided for legacy clients that cannot send a username. ■ <AUTO> is provided for site-specific credentials.
<password>	Password for the SDK user requesting a connection.	Can be a password or one of the following special values: ■ <SYS> indicates that the SDK username must be a valid login name on the local system. The server passes the credentials to the host operating system for validation. ■ <DEFAULT> is provided for legacy clients that cannot send a password. ■ <AUTO> is provided for site-specific credentials.
<privilege>	Access privileges of the client.	Valid values include: ■ All ■ Monitor ■ None ■ Ping

Remember that during the authentication process the server receives a connection target, SDK username, and password from a client. The server checks each of its records looking for a match. When it finds the first match, it sends the appropriate privilege back to the client. Otherwise, the server logs the failed authentication.

Connection privileges

Connection privileges are specified in the serverConnect.conf file. SDK authentication features provide for four levels of privileges:

- *All* provides full access to all server functions. It is required for all adapter-to-server and server-to-server connections and by all command line utilities, with a few exceptions. This level of authorization is also required by administrative consoles. However, this level of access can be further restricted through user profiles. The
  VMware Smart Assurance Service Assurance Manager Configuration Guide provides additional information about using user profiles to restrict access to console operations.

- *Monitor* access supports a monitoring console. A monitoring console cannot change server database or configuration parameters except in special circumstances such as acknowledgement of notifications. Only consoles support Monitor access. If you run a secure Broker, the Broker must also support ping (<BROKER>:BrokerNonsecure:Nonsecure:ping).
- *None* prevents access to the server. This privilege can be used to explicitly prevent an SDK user from accessing the server.
- *Ping* is a special access privilege that allows connections to a server, but only to ping the server and check whether it is functioning. A Broker requires Ping access to check the status of servers and is sufficient to allow the dmctl command line utility to connect to a server and execute the ping command.

Client authentication records

Most clients connect to servers without requiring a user to type an SDK username or password. When this happens, a client parses a configuration file for a username and password to send to the server for authentication instead of prompting a user. Most clients use the clientConnect.conf. The Broker uses its own file, brokerConnect.conf.

Note The Broker only uses brokerConnect.conf to send authentication to other processes so that it can ping them.

Description of clientConnect.conf and brokerConnect.conf

By default, these files are located in the BASEDIR/smarts/conf directory. The sm_edit utility saves changes to the file in BASEDIR/smarts/local/conf and does not modify the original version of the file.

The format of a record in the clientConnect.conf or brokerConnect.conf is:

```
<login user>:<target>:<
            SDK

          username>:<password>
```

Fields for clientConnect.conf and brokerConnect.conf describes the four fields of an authentication record in the clientConnect.conf and brokerConnect.conf files.

Table 5-3. Fields for clientConnect.conf and brokerConnect.conf

Field	Definition	Value
<login user>	System login name of the person or process attempting a connection.	Can be a matching pattern with wildcards.
<target >	Name of the server to which the client is trying to connect.	Can be a matching pattern with wildcards or one of the following special values: ■ <BROKER> indicates that this record applies only to the Broker. ■ ~<BROKER> indicates this record applies to all servers except the Broker.

Table 5-3. Fields for clientConnect.conf and brokerConnect.conf (Continued)

Field	Definition	Value
<SDK username>	SDK username that is sent to server for authentication.	Can be a username or one of the following special values: ■ <USERNAME> indicates that the username under which the current process is logged in as is sent as the SDK username. ■ <PROMPT> indicates that the client program asks the user to provide an SDK username. ■ <AUTO> is provided for site-specific credentials. ■ <DEFAULT> mimics legacy client authentication.
<password>	Password that is sent to the server for authentication.	Can be a password or one of the following special values: ■ <PROMPT> indicates that the client program asks the user to provide a password. ■ <AUTO> is provided for site-specific credentials. ■ <DEFAULT> mimics legacy client authentication.

It is important to remember that an SDK program runs under the login name of the user who started it. This has several implications:

- A user account must provide sufficient privileges for the program to function properly. For example, a Manager may need to run with root privileges because it sends ICMP pings or receives SNMP traps.
- A Domain Manager is as a Broker, and the Broker is listening on port 426. Port 426 is a privileged port, meaning the process must be running as root to open the port. Ports numbered below 1024 require root privileges.

  Note A process, without being root, can connect to a process listening on a port below 1024.

- A user's system login name must correspond to an SDK username in the clientConnect.conf file or an SDK username and password will not be sent to a server for authentication. In the clientConnect.conf record, a user's login name and SDK username do not have to be identical.

Running without root privileges

All Domain Managers do not need to send ICMP pings or receive SNMP traps and, therefore, do not need to run with root privileges.

- Start a Broker on a port that is not privileged. VMware software registers the following non privileged ports:
  - Broker is port 426
  - Domain Manager is port 4426

    The Broker could be started on port 4426 if a well-known port is required.
- Domain Managers can startup as root, open privileged ports, and then change to a non root user, as described in Running as non root on UNIX and Linux systems.

Special authentication values

Special authentication describes the configuration of the serverConnect.conf, clientConnect.conf and brokerConnect.conf by using the special values <SYS>, <PROMPT>, <AUTO>, and <DEFAULT>. These values control system, site-specific and legacy authentication.

System authentication

System authentication uses the operating system username and password to authenticate clients. This method is configured in serverConnect.conf. By using this method, you give every console operator an account on the host on which the server runs. The operators log in by using the username and password defined for that account. (The account can be disabled to prevent actual interactive access to the system.) By using this method:

- Each console operator has a unique username and password.
- Accesses to the system can be traced to a particular user.
- Access can be individually revoked.

  The use of <SYS> for the password allows the use of common password administration across applications and avoids having the password appear in cleartext in the file in unencrypted installations.

  This mechanism can readily be extended to provide similar controlled access for administrators. For example, you could add the following records to serverConnect.conf:

  ```
  * : fred|george : <SYS> : All
  * : * : <SYS> : Monitor
  ```

  This example would grant the users “fred” and “george” All access, once they provide the passwords for their accounts on the host. Other users providing the correct password are granted Monitor access.

  You could even define a class of administrative users, for example, with usernames that start with ADM and provide all other users with Monitor access.

  ```
  * : ADM* : <SYS> : All
  * : * : <SYS> : Monitor
  ```

  Console applications automatically prompt for a username and password and do not use clientConnect.conf. In console applications, passwords display as masked characters (*) to avoid displaying the passwords in cleartext. In order for nonconsole applications to prompt, the value <PROMPT> must be used for the SDK username and password in clientConnect.conf.

Network account authentication

In UNIX systems, the system authenticates a username based on its password.

Prompting for authentication

In order for nonconsole applications to prompt for a username and password, the corresponding fields in clientConnect.conf must contain <PROMPT>. The client must be attached to a terminal, in order for the system to prompt the user. Lines containing a <PROMPT> are skipped by programs that are not attached to a terminal, even if they would otherwise be selected.

Responses to <PROMPT> are checked against username and password fields in serverConnect.conf.

Site-specific authentication

Site-specific authentication uses a site's secret to authenticate connections. When corresponding records in serverConnect.conf and clientConnect.conf contain the <AUTO> value for both the SDK username and password, the client generates a password by using its secret. The server validates the password to accept or reject the connection.

Note Do not use this method unless you have changed the default site secret. The default site secret is known to all SDK installations.

So for connections to a particular Domain Manager (such as Service Assurance Manager), you could define a record in serverConnect.conf:

```
GM-Company-1:<AUTO>:<AUTO>:All
```

Then define a corresponding record in the clientConnect.conf files on the different hosts connecting to the server:

```
*:GM-Company-1:<AUTO>:<AUTO>
```

As long as both the client and server use the same secret phrase, the server will authenticate each connection to the Domain Manager.

Note Do not use <AUTO> for the brokerConnect.conf if it is used for other connections, because the Broker does not need All access, only Ping.

Legacy system authentication

To provide the ability to interoperate with older SDK software, which may not support authentication, SDK software provides a "default account" mechanism. When an incoming connection does not provide any authentication information, a server substitutes the standard username and password with <DEFAULT>. After that substitution, the <DEFAULT>/<DEFAULT> authentication information is validated in exactly the same way any other username/password combination is validated.

An incoming connection that explicitly specifies <DEFAULT>/<DEFAULT> is permitted. It is treated in exactly the same way as a connection that supplied no authentication information.

Default authentication records

This section describes the authentication records enabled by default. These default settings provide examples of user records in serverConnect.conf as well as examples of how automatic authentication can be set up in clientConnect.conf and brokerConnect.conf.

Authenticating legacy software

The Broker uses the following record in serverConnect.conf to authenticate legacy client connections and provide full access. The target field, with a value of <BROKER>, identifies that the Broker is the sole target for authentication. There are no corresponding entries in the clientConnect.conf or brokerConnect.conf. Legacy system authentication provides additional information on the <DEFAULT> value.

```
<BROKER> : <DEFAULT> : <DEFAULT> : All
```

In this configuration, the Global Console will not prompt for a username and password when connecting to the Broker. The default configuration defines a nonsecure Broker, equivalent in security to Ionix software prior to version 5.0.

Automating client authentication to the Broker

SDK programs connect to the Broker under the following conditions:

- When a server starts, it connects to the Broker as a client to register itself.
- When a client, including a console, needs to connect to a server, it connects to the Broker to query it for the server's location.

  The record in serverConnect.conf specifies that the Broker should grant any client that sends the SDK username BrokerNonsecure and the password Nonsecure full access to the Broker.

  ```
  <BROKER> : BrokerNonsecure : Nonsecure : All
  ```

  The automatic authentication record in clientConnect.conf applies to any client that supports authentication and needs to connect to the Broker. This record specifically selects the Broker as its target.

  ```
  * : <BROKER> :BrokerNonsecure : Nonsecure
  ```

  These two records define a nonsecure configuration for the Broker. In this configuration, consoles do not prompt for a username and password when connecting to the Broker. *"Configuring a secure Broker" on page 108* provides additional information regarding a secure Broker configuration.

Automating Broker authentication to servers

The Broker periodically pings all registered servers to determine their status. When the Broker does this, it acts as a client. This record in serverConnect.conf permits the Broker to ping a server to check if it is running.

```
* : BrokerPing : Ping : Ping
```

The brokerConnect.conf file contains an automatic authentication record, the only record in the file.

```
* : * : BrokerPing : Ping
```

Connection privileges provides additional information about the Ping access permission.

Authenticating administrative users

The default administrative record provides full access to any server by using the serverConnect.conf file.

```
* : admin : changeme : All
```

This record authenticates clients that provide the SDK username “admin” and the password “changeme”. This account grants administrative privileges or full access, which is denoted by the value of **All** in the privilege field.

Note Change the password for this account after installation. You must make any changes to the corresponding records in both the serverConnect.conf and clientConnect.conf files. Replace the admin user only *after* you have set up a corresponding administrative account in the Global Manager. You should *never* remove or disable an administrative account in the Global Manager.

The corresponding automatic authentication record in clientConnect.conf grants any client, which uses the file, full access to any server by using the corresponding record in the serverConnect.conf file.

```
* : * : admin : changeme
```

Since this record appears later in the clientConnect.conf file than the interactive users record (* : * : <PROMPT> : <PROMPT>), only noninteractive clients will be granted full access, unless interactive users type the correct username ("admin") and password ("changeme").

Connection privileges provides additional information about the All access permission.

Authenticating Global Console users

Two records in serverConnect.conf provide the console with monitoring access to servers. The first authenticates console users who provide the SDK username maint and the password maint.

```
* : maint : maint : Monitor
```

The second console record authenticates console users who provide the SDK username oper and the password oper.

```
* : oper : oper : Monitor
```

Consoles do not use clientConnect.conf for authentication (there is no automatic authentication). The console prompts the user for a username and password and passes the information to the server for authentication.

Note User profile restrictions can be used to further limit certain Global Console operations, as described in the VMware Smart Assurance Service Assurance Manager Configuration Guide.

VMware recommends that you create separate accounts for each operator. Comment out these records and use the authentication record that uses the system login facility.

Authenticating users with a valid system account

The last default authentication record in serverConnect.conf permits users with a valid login account on the system where the server is running to connect. A client that uses this authentication record must type a valid system username and password. For authentication, the system username becomes the SDK username. This record provides monitoring privileges.

```
* : * : <SYS> : Monitor
```

Authenticating users interactively

This authentication record in the clientConnect.conf prompts users to type an SDK username and password. This does not guarantee that the connection will be successful. The server must be able to validate the username and password by using its serverConnect.conf file.

```
* : * : <PROMPT> : <PROMPT>
```

The position of this authentication record in the default clientConnect.conf file is important. Because it uses wildcard patterns to match for both login user and target, this record is always selected for interactive connections, even if a following record also matches. Noninteractive programs skip this record because they cannot prompt for information.

This record could match the following serverConnect.conf authentication records, as described in the followingAuthenticating users with a valid system account:

```
* : * : <SYS> : Monitor
* : maint : maint : Monitor
* : oper : oper : Monitor

* : * : <SYS> : Monitor
```

Specifying alternate security configuration files

You can define separate serverConnect.conf and clientConnect.conf files on hosts where multiple servers or clients are running. This enables you to configure a system where certain users and/or servers use one file, and other users and/or servers reference a different file. Some installations can share the same BASEDIR/smarts, but have requirements that stipulate that the servers and/or clients operate differently.

Two environment variables allow you to specify distinct configuration files:

- SM_SERVERCONNECT
- SM_CLIENTCONNECT

  For each variable, the absolute path to the alternate security file must be specified. Because you specify the filename, you can have multiple files in the same directory.

  For server programs, you can specify an alternate serverConnect.conf file, by setting the SM_SERVERCONNECT variable in the command line that starts the server.

  For example, the following steps set the value of the SM_SERVERCONNECT environment variable to point to the server_Connect_IP.conf file.

  a Use the **sm_service show** action with the **--**cmdlineoption to display the existing command line for the program. You must have administrative privileges when typing the following command.

```
                        BASEDIR/smarts
                     /bin/sm_service show --cmdline ic-am-server


                        BASEDIR/smarts
                     /bin/sm_service install
 --startmode=runonce ic-am-server

                        BASEDIR/smarts
                     /bin/sm_server
        --name=INCHARGE-AM
        --config=icf
        --bootstrap=bootstrap-am.conf
        --port=0
        --ignore-restore-errors
        --output
```

  b Use the **sm_service install** action with the **--**forceoption to add the environment variable to the command line. The **--**envoption specifies the environment variable.

```
t
                        BASEDIR/smarts
                     /bin/sm_service install --force
 --startmode=runonce
 --env=SM_SERVERCONNECT=
                        BASEDIR/smarts
```

```
                    /conf/
 serverConnect_IP.conf
ic-am-server

                       BASEDIR/smarts
                    /bin/sm_server
       --name=INCHARGE-AM
       --config=icf
       --bootstrap=bootstrap-am.conf
       --port=0
       --ignore-restore-errors
       --output s
```

c Do one of the following:

- Start the program.
- Stop and restart the program.

  For client programs, you can specify an alternate clientConnect.conf file by setting the SM_CLIENTCONNECT variable in the runcmd_env.sh file.

Controlling authentication

The environment variable SM_AUTHORITY controls the authentication security features provided by SDK software. The authentication features are enabled by default. If necessary, you can set the value of this variable to disable them. This environment variable must be set on each system where SDK software is running.

The environment variable SM_AUTHORITY can have one of two values:

- <STD> enables security
- <NONE> disables security

  When no value is specified, which is the default, SDK software treats this the same as <STD>.

  Note The angle brackets (< >) are required.

  When SM_AUTHORITY is set to <NONE>, SDK software behaves as if no security features are in effect:

- Clients do not read clientConnect.conf and never prompt for a username or password. They always send <DEFAULT>/<DEFAULT> as their credentials.
- Servers ignore any incoming username or password and grant any connection All access.

Setting the SM_AUTHORITY variable

To add and set the SM_AUTHORITY environment variable in the runcmd_env.sh file in the BASEDIR/smarts/local/conf directory:

1 Use sm_edit to open the runcmd_env.sh file.

```
sm_edit conf/runcmd_env.sh
```

2 Add the environment variable and its value by using the following syntax:

```
SM_AUTHORITY="<STD>"
```

3 Save the runcmd_env.sh file and close it.

Any SDK program started after this point will use the applicable environment variables specified in the runcmd_env.sh file. SDK programs that are already running need to be restarted for any new environment variable to take effect.

Encryption

By default, encryption is enabled during the installation process. The basis for encryption is a secret phrase that gets transformed into the file imk.dat. The SDK programs use this site secret to encrypt passwords in the configuration files and passwords to SNMP v3 devices in seed files, as well as to encrypt connections between SDK programs.

At installation, encryption is enabled with a default secret phrase. This phrase is:

```
Not a secret (literally)
```

and the imk.dat file can be copied. The imk.dat is found in the BASEDIR/smarts/local/conf directory.

Important The imk.dat file can only be copied to other systems with the same operating system (OS) and OS version.

To raise the level of security, you need to change the secret phrase by using the sm_rebond utility. Thereafter, the secret phrase should be changed periodically to maintain a secure system.

Note The secret phrase should be treated with the same care as a root password or highest level system administration password.

Changing the secret

The sm_rebond utility changes the secret phrase and re-encrypts the files affected by the secret. This utility prompts for the old secret phrase and the new phrase, then generates an imk.dat file and updates all the files containing encrypted information. This utility affects all of the applications that run on the same host and use the same imk.dat file.

Note The sm_rebond utility shuts down all of the processes run from the suite that were started by using sm_service or sm_serviced, and that use the same imk.dat file. The sm_rebond utility re-encrypts the security files and the seed files, then restarts the processes once the phrase and encryption changes are made. All other processes should be shut down manually before running the utility and restarted once the utility is finished.

To change the secret, perform the following steps:

1 Use the sm_rebond utility. You must type this command with administrative privileges.

```
sm_rebond --basedir=<base directory>
```

2 When the utility prompts for the old secret phrase, type it.

3 When the utility prompts for the new secret phase, type it. The secret phrase can consist of a combination of printable characters, integers and special characters. The secret phrase cannot be more than 1,024 characters long.

4 Retype the new secret to confirm it.

5 Close the utility.

Note The sm_rebond utility encrypts only files that reside in BASEDIR/smarts/local/conf and three levels of subdirectories below that. To encrypt files outside of that directory area, contact VMware Global Services.

Using the secret for multiple suites and sites

The secret phrase can be used across multiple suites and sites. To use the secret phrase across suites and sites, VMware, Inc. recommends that you perform the following steps:

1 Install one or more SDK suites on one or more hosts.

2 If short periods of interrupted communications are not detrimental, stop the Managers. However, if the suites also use cleartext connections, you can use the sm_rebond utility without stopping the Managers. Only that suite will lose connections for a short period of time.

3 Use the sm_rebond utility to change the secret phrase for each suite on each host.

Note The secret phrase must be exactly the same for each suite on each host in order for the applications to make connections.

4 Use the sm_edit utility to update the security configuration files and seed files of the suites as needed.

5 Restart the Managers if applicable.

Using the secret across large numbers of sites

For large numbers of sites, you can perform the following steps:

1 Install an SDK suite at the primary site.

2 Use the sm_rebond utility to change the secret phrase.

3 Use the sm_edit utility to update the security configuration files and seed files of the suite as needed.

4 Copy the imk.dat file, as well as the configuration files and seed files to the other installations of the SDK software.

Important The imk.dat file can only be copied to other systems with the same operating system (OS) and OS version.

You will not have to use the sm_rebond utility on the other Managers if the files are moved to the Managers before the Managers are started.

Locking the secret

If access to encrypted data is a critical issue for your organization, VMware provides the option to lock the imk.dat file. This disables copies of the imk.dat file, even by restoration from a backup process. For more information about this level of security, contact VMware Global Services.

Encrypted passwords

The encryption process uses the secret to encrypt passwords contained in the security configuration files. The first line of the serverConnect.conf, clientConnect.conf, and brokerConnect.conf files indicates the appropriate field to which encryption is applied.

The first line in the file must read as follows. It must begin with a pound sign (#).

```
#<encrypted field>:1.0:<n>
```

The value n refers to the position of the Password field in the configuration file's record and should not be changed:

- For serverConnect.conf, this value is 3.
- For clientConnect.conf and brokerConnect.conf, this value is 4.

  To disable encryption, add a second pound sign (#) at the beginning of the line.

Note Installation encrypts the password fields in the files by using the default secret. The encrypted files reside in BASEDIR/smarts/local/conf.

Encrypting passwords

Password is the only field in the security configuration files that is encrypted. The configuration files are located in the BASEDIR/smarts/local/conf directory of a product suite.

To type or change a password in a configuration file:

1 Use sm_edit to open the file.

For example, to open the serverConnect.conf file, invoke sm_edit from the BASEDIR/smarts/bin directory:

```
sm_edit conf/serverConnect.conf
```

2 Review the contents of the file. For example, the serverConnect.conf file may include lines similar to the following:

```
#
<BROKER>    :<DEFAULT>:<DEFAULT>:All
InCharge    :admin    :<SYS>    :All
InCharge-WP     :user5    :Password :Ping
InCharge    :operator :Password :Monitor
InCharge    :*        :<SYS>    :None
#
```

3 Delete the current password and replace it with the new password in cleartext. If the current value is encrypted, delete the **<E-1.0>** tag that marks the password as encrypted as well as the current password. The special values <DEFAULT>, <PROMPT>, <SYS>, and <AUTO> in the files are not encrypted.

4 Save the file and close the file.

The appropriate values are encrypted as part of the process.

5 Repeat the steps for all of the configuration files.

If a configuration file already includes the first line for encryption, specify the sm_edit utility with the --noedit option to encrypt the file without opening it. You must have administrative privileges when typing this command.

```
BASEDIR/smarts

/local/conf/sm_edit --noedit <filename>
```

Encrypted seed files

The encryption for a seed file only applies to certain information that applies to SNMP v3 devices. For these devices, a keyword that contains the authentication password can be encrypted.

The first line of the seed file must read as follows; it must begin with a pound sign (#).

```
#<encrypted seed>:1.0:AUTHPASS,PRIVPASS
```

The AUTHPASS and PRIVPASS keywords specify the authentication and privacy passwords for an SNMP v3 device.

To disable encryption, add a second pound sign (#) at the beginning of the line.

If a seed file already includes the first line for encryption, specify the sm_edit utility with the --noedit option to encrypt the file without opening it. You must have administrative privileges when typing this command.

```
BASEDIR/smarts

/local/conf/sm_edit --noedit <filename>
```

Encrypted connections

SDK components communicate over TCP connections by using the Ionix Remote API. Clients which are using Remote API connections, authenticate themselves to servers by sending credentials, nominally a username and password. When the credentials are passed as cleartext, they can be snooped from the network or accessed by using man-in-the-middle configurations.

You can encrypt certain connections by using different keys for the Advanced Encryption Standard based on a combination of the Diffie Helman standard and the site secret associated with the SDK installation. *"Encryption" on page 100* provides additional information about the site secret.

Encrypted connections do not work with the following products:

- Perl API
- VMware Smart Assurance Adapter for NetIQ AppManager

Levels of encryption

SDK software provides three levels of security above cleartext communication: Diffie Helman-Advanced Encryption Standard (DH-AES), encryption based on the site secret, and DH-AES used in conjunction with the site secret.

Any encryption based on the site secret should only be used once the secret phrase has been changed by using sm_rebond described in Changing the secret.

Encryption levels for connections lists the four types of encryption connections.

Table 5-4. Encryption levels for connections

Security level	Description	Advantages	Disadvantages
0, CLEAR, or CLEARTEXT	no encrypted communication	Backwards compatibility, no configuration (default behavior)	No security, passwords passed to servers as cleartext
1	DH-AES	No site secret needed, no configuration (default behavior for new installations), protects against eavesdroppers	Slower connection than cleartext or level 2 security, not secure against active attacks
2	Encryption based on site secret	Protects against eavesdropping and active attack, almost as fast as cleartext	Must set site secret and keep it common across all communicating entities
3	DH-AES and site secret	Protects against eavesdropping and active attack, even by those who know the site secret	Slower connection than cleartext or level 2 security, must set site secret and keep it common across all communicating entities

Note The Global Console supports both cleartext (Level 0) and Diffie Helman-Advanced Encryption Standard (Level 1) encrypted connections.

Encrypting connections

Encrypted connections are configured by using two environment variables:

- SM_OUTGOING_PROTOCOL— Controls the connections that an SDK program acting as a client is allowed to initiate.
- SM_INCOMING_PROTOCOL— Controls the connections that an SDK program acting as a server is allowed to accept.

  Each of these can contain a list of security levels. If you specify more than one, separate them with commas. If you do not set the variable, it is the same as specifying “0”, or CLEARTEXT.

  When two or more security levels are supported in common, the largest level number is used. For example, if a client has SM_OUTGOING_PROTOCOL set to "0,1" and the server has SM_INCOMING_PROTOCOL set to "0,1,2", the connection uses the highest common level, level 1.

  The order in which the different levels are specified for the environment variables has no significance —for example, "0,1" and "1,0" behave exactly the same.

  Applications that do not support encryption are treated as only having a cleartext level of encryption.

  When the two ends of the connection do not match, such as when SM_OUTGOING_PROTOCOL is 3 at server A and SM_INCOMING_PROTOCOL is 2 at server B, a connection cannot be formed. Both client and server will report errors.

To set the environment variables so that they can be used by the programs of an SDK product suite, edit the runcmd_env.sh file, which is located in the BASEDIR/smarts/local/conf directory of that product suite.

a Use sm_edit to open the runcmd_env.sh file. Invoke sm_edit from the BASEDIR/smarts/bin directory:

```
sm_edit conf/runcmd_env.sh
C:\>sm_edit conf\runcmd_env.sh
```

b Review the contents of the file. The default values for the SM_INCOMING_PROTOCOL and SM_OUTGOING PROTOCOL are 1 and 0.

```
SM_INCOMING_PROTOCOL=1,0
SM_OUTGOING_PROTOCOL=1,0
```

c Update the values for the variables to meet the needs of your system.

For example, to raise security to the next highest level, change **1** to **2**:

```
SM_INCOMING_PROTOCOL=2,0
SM_OUTGOING_PROTOCOL=2,0
```

d Save the runcmd_env.sh file and close it.

e Any SDK program within the product suite started after this point will use the environment variables specified in the runcmd_env.sh file. SDK programs that are already running need to be restarted for any new environment variable to take effect.

Suggested encrypted connections

VMware recommends that you configure your system to use encrypted connections wherever possible.

- Brokers should be able to communicate with cleartext as well as encryption in both SM_INCOMING_PROTOCOL and SM_OUTGOING_PROTOCOL if a client only supports cleartext. This is required in this configuration since the Broker acts as both a client and a server, and must be able to communicate with every component in the system. Brokers do not need to support cleartext if all clients can make encrypted connections.
- If a Domain Manager must connect to a client that only supports cleartext, then set SM_OUTGOING_PROTOCOL to cleartext as well as encryption.
- Configure adapters with SM_OUTGOING_PROTOCOL set to require encryption. Only adapters that register with the Broker (**--**name option) can accept incoming connections. If you have adapters that accept incoming connections, setting SM_INCOMING_PROTOCOL to require encryption is appropriate.

  Also, if the adapter must connect, or be connected, to clients that support only cleartext, then add the cleartext option to the appropriate variable.

- Configure any components that must run on networks outside the management domain with both SM_INCOMING_PROTOCOL and SM_OUTGOING_PROTOCOL set to encryption. Depending on the level of encryption, this will prevent snooping or man-in-the-middle attackers. You will not be able to connect directly to such a component by using a console.

Configuring a secure Broker

You can configure the Broker to run in a secure manner. Use of a secure Broker results in the following changes to how SDK software runs:

- Consoles prompt for a username and password to connect to the Broker. Without a secure Broker, consoles connect to the Broker without authenticating.
- Other servers and clients use their respective clientConnect.conf files to determine what credentials to send to the Broker, just as they use clientConnect.conf to determine what credentials to send to a server. In particular, you can configure the clientConnect.conf files so that clients and servers prompt for connections to the Broker, as the console does, or specify the password in clientConnect.conf.

  To configure and run a secure Broker, complete the following steps:

  a Choose a unique SDK username and password for the secure Broker credentials. The new SDK username and password will be used by both servers and clients:

    - Servers will use these credentials to register with the Broker.
    - Clients will use these credentials to connect to the Broker and determine the location of a server.

      For example, you could use the SDK username “SecureBroker” and the password “Secure”. Choose a unique SDK username and password.

  b Use the sm_edit utility to open a local copy of the clientConnect.conf file, located in BASEDIR/smarts/local/conf. Edit this file, used by all clients and servers, so that SDK programs send the SecureBroker/Secure credentials when connecting to the Broker.

    - Comment out the following line:

      ```
      *:<BROKER>:BrokerNonsecure:Nonsecure
      ```

    - Type a new line configuring a secure Broker. This new line is added below the BrokerNonsecure line that you commented out.

      For example:

      ```
      #*:<BROKER>:BrokerNonsecure:Nonsecure
      *: <BROKER> : SecureBroker : Secure
      *: <BROKER> : SecureBroker : <PROMPT>
      ```

  c Use sm_edit to make the following changes to the local serverConnect.conf file used by the Broker:

    - Delete the line granting <DEFAULT>/<DEFAULT> access to the Broker.

- Change the BrokerNonsecure/Nonsecure line to grant Ping access rather than All access. Do not, however, delete this authentication record.
- Add a new authentication record that grants All access to the SecureBroker/Secure credentials This new record must be below the BrokerNonsecure/Nonsecure record. For example:

```
<BROKER>:BrokerNonsecure:Nonsecure:Ping
<BROKER> : SecureBroker : Secure : All
```

Additional safeguards

In addition to encrypted passwords and encrypted connections, you can secure your system by using the file permissions of the operating system, by limiting access to the security configuration files, and by limiting access to the servers.

Using file permissions

You can limit access to the security configuration files by using the permissions feature of the operating system. Only select users should be allowed access to the files. The users must include those who launch the SDK applications requiring access to the authentication records.

Limiting access to the configuration files

You should configure security in such a way that each security file is only readable by those programs or users that require it. The security configuration files installed with SDK software, which should be edited after installation, are readable by anyone.

Because Managers and the Broker typically run with administrative privileges, the serverConnect.conf and brokerConnect.conf files should only be readable by users with administrative privileges and no one else.

The simplest method for creating a secure setup for users and client programs is to provide two separate clientConnect.conf files. One clientConnect.conf file, which can remain readable by anyone, should only contain entries that make client programs prompt for passwords. This clientConnect.conf file will not contain passwords.

For client programs, create a separate clientConnect.conf file that contains the authentication information necessary for non-prompting programs to access Managers. This clientConnect.conf should only be readable by the user(s) under which these programs run. Client programs use the SM_CLIENTCONNECT environment variable to find this clientConnect.conf file. You can specify SM_CLIENTCONNECT in the service startup file for each service. For clients that are installed as services, you can use the **--env** option to the sm_service utility to edit the parameters of a service.

Limiting access to servers

The **--**acceptoption to the sm_server command, described in SM_SNMP_BUG_COMPATIBLE, provides another method of access control. This option limits the hosts that can connect with the server. Before other security measures occur, incoming connections must pass the **--**accept option before authenticating.

Example security configurations

You can configure SDK installations to use a variety of levels of security. Higher levels of security require more administrative effort, as well as extra effort on the part of users of the system, but give you better control of the use of SDK software. The appropriate level of security for your site is determined by the policies and procedures of your organization.

This section describes two levels of security, and their advantages and disadvantages. You can choose one of these levels, or combine characteristics of several.

Default security

When initially installed, default security uses the security configuration files with encrypted passwords based on the default secret phrase. The connection protocols are set to use Diffie Helman encryption and cleartext. With the default security, you need to change the passwords for the SDK usernames in the security configuration files. Default security prevents casual inspection, but does not pose much of a deterrence from more strenuous efforts.

Note It is essential that you change the default passwords for the usernames before using the system in a production environment.

While the default security provides basic authentication, it does not provide much manageability or security. The oper/oper credentials must be known to all console operators; the admin/changeme credentials must be known to all administrators. This makes it difficult to know who is actually connected to a server, based on their username. In addition, you cannot revoke the rights of one user without changing the SDK username and password for all users at the same access level.

The serverConnect.conf and clientConnect.conf files include authentication records that improves user administration. First, besides the oper/oper credentials, there are maint/maint credentials that also provide Monitor access. This illustrates how one might define shared usernames that are still differentiated on the basis of role. Second, serverConnect.conf contains the following line:

```
* : * : <SYS> : Monitor
```

Anyone who can provide a username and password that the operating system considers valid is also granted Monitor access. In this scenario, you can give every console operator an account on the host on which the server they access runs. These operators log in by using the username and password defined for that account. The account can be disabled to prevent actual interactive access to the system. In this way, each console operator gets a unique username and password. Accesses to the system can be traced to a particular user, and access can be individually revoked. Specifying <SYS> for the password avoids having to potentially create a number of records in serverConnect.conf.

This mechanism can readily be extended to provide similar controlled access for administrators. For example, you could add the following records to serverConnect.conf:

```
* : fred|george : <SYS> : All
* : * : <SYS> : Monitor
```

This would give the users “fred” and “george”, if they provide the passwords for their accounts on the host, All access. You could even define a class of administrative users, for example, with usernames that start with ADM.

```
* : ADM* : <SYS> : All
* : * : <SYS> : Monitor
```

Customized security

Customized security uses security files (with encrypted passwords), a new secret phrase, and passwords for the SDK usernames in the security files that are different than their default values. Under customized security, connection protocols are set to use the site secret and cleartext. This setup protects against eavesdropping and active attacks as well as casual inspection.

To configure customized security:

1 Create a new secret phrase by using sm_rebond, as described in Changing the secret.

2 Change the username and passwords in serverConnect.conf, clientConnect.conf, and brokerConnect.conf.

   These should provide access control by individual users. Consider using system authentication to authenticate Global Console users, as described in System authentication. The new usernames and passwords should also address connections between servers and servers or servers and clients.

   If your SDK software resides on a single host, a single username and password for automatic client authentication is usually sufficient. However, if you have clients, such as adapters, on multiple hosts, or generally in different security domains, you may use different usernames and passwords for manageability. For example, if there is reason to believe that the username and password from one machine have been compromised, you can change them without having to change the configuration on any other machine.

To achieve such a configuration, you need to assign unique usernames and passwords to clients that you consider to reside in separate security domains. Add a line to grant access to each such username/password credential to the serverConnect.conf file of each server the client will access. Add a corresponding line to the clientConnect.conf file that the client will use.

Note If you have multiple security domains on a single host, use the SM_CLIENTCONNECT environment variable so that different clients read different clientConnect.conf files. You can use the **--**env option of the sm_service utility to set the environment value when the service starts.

3 Configure the different SDK components to communicate with encrypted connections. The types of allowed connections depend on the versions of SDK software in use.

- Most connections should be encrypted. The Broker does not need to support cleartext if all clients can make encrypted connections.

  Encrypted connections on page 105 provides additional information.

4 At the highest security level, you should run with a secure Broker as well. Setting up a secure Broker is described in *"Configuring a secure Broker" on page 108*.

Operation of the Broker

6

This chapter includes the following topics:

- Overview of the Broker
- Viewing the registry of the Broker
- How the SDK clients find the Broker
- How to change the Broker environment variable
- Securing the Broker

Overview of the Broker

An SDK client application, such as a console or an adapter, utilizes the Broker to determine where Managers are running. When a Manager starts, it registers with the broker:

- Hostname of the machine it is running on
- TCP port it is listening on.

  SDK clients retrieve this information from the Broker so that they can communicate with the Manager.

  The Broker registry maintains the following information:

- Name of the Manager, including the host and TCP port it is running on.
- Status of each Manager.

  The Broker checks the status of each Manager every five minutes by connecting to the host on which SDK is running and determining if the SDK process is running properly. If the Broker is unable to connect or the process is not running, the Broker changes the status of the Manager to Dead.

  - *Running* indicates that the Broker is able to communicate with the Manager.
  - *Dead* indicates that the Manager exited unexpectedly or is unreachable. When a Manager properly shuts down, it notifies the Broker and the Broker removes it from its registry.
  - *Unknown* indicates that the Broker was restarted and that it is querying its list of Managers to determine their state.

- Process ID of each Manager. This is the process ID assigned by the host's operating system. In some cases when the Broker is restarted, the process ID of each Manager is set to zero to indicate the Broker does not know the process ID of the Manager.

- Last time the state of the Manager changed. This value is set when the Manager registers with the Broker and is updated if the Broker determines that the Manager is dead. When the Broker restarts, it changes the status of Managers marked Unknown to Running or Dead.

  The Broker changes the status of the Manager to Dead when it cannot connect to the Manager. However, the Broker continues to try to connect to the Manager every five minutes. If the Broker succeeds in connecting to the Manager, it changes its status back to Running.

Viewing the registry of the Broker

The Broker is a component of an SDK application. The Broker is designed to automatically start each time the host on which it is installed is started.

You can use the brcontrol utility to view the list of registered programs.

Broker command line option

To view the contents of the Broker from a command prompt, use the following command:

```
   BASEDIR/smarts
/bin/brcontrol [-b <host>:<port>]
```

If the Broker resides on another system, specify the **-**b option with the host and port number.

How the SDK clients find the Broker

The Broker facilitates communication between SDK clients and Managers. During installation, you are prompted for the host location of the Broker and its port number. This information is stored in the SM_BROKER_DEFAULT environment variable. This variable is used by all SDK programs to find the Broker. The default host setting for the variable is *localhost*, and the default port setting is 426.

An SDK client performs the following steps to determine the Broker's location.

1. Verifies whether the Broker's location was specified as an option at startup. When this option is specified, no other options are checked.
2. Checks the value of the SM_BROKER_DEFAULT environment variable. If this variable is set, no other options are checked.
3. Checks if the Broker is running on the host smarts-broker and listening on TCP port 426.
4. If smarts-broker is not defined, the program checks port 426 on localhost.

Note The hostname smarts-broker is usually an alias, such as a DNS CNAME.

How to change the Broker environment variable

The default location of the Broker is specified at installation and stored in the SM_BROKER_DEFAULT environment variable.

There are two scenarios in which you may need to change the value of the environment variable after SDK software is installed.

- If during installation, the default Broker host and/or port settings were changed, but the typed values are incorrect.
- If the Broker is automatically installed when you install an SDK application or the adapters. Because only one Broker should be active on a network, the Broker on one of the hosts must be disabled, and the Broker environment variable on that host must point to the host where the Broker is running. This typically occurs when an SDK application is installed on one host and the SDK adapters are installed on a second host.

Correcting the Broker settings

In the first scenario, you must edit the Broker host and/or port value in two places.

1 First, change the SM_BROKER_DEFAULT environment variable in the runcmd_env.sh file, which is located in the BASEDIR/smarts/local/conf directory:

```
SM_BROKER_DEFAULT=<host>:<value>
```

2 Additionally, you must change the **--**port value in the **sm_service install** Broker command line to match the port value you set in the SM_BROKER_DEFAULT environment variable.

Use the **sm_service show** action with the **--**cmdline option to display the existing install Broker command line.

```
                 BASEDIR/smarts
              /bin/sm_service show --cmdline ic-broker



                 BASEDIR/smarts
              /bin/sm_service install
--startmode=runonce
--env=SM_CLIENTCONNECT=brokerConnect.conf ic-broker

                 BASEDIR/smarts
              /bin/brstart
--port=<PORT>
--output
--restore=
                 BASEDIR/smarts
              /local/repos/broker/broker.rps
```

```
t
                   BASEDIR/smarts
               /bin/sm_service install --force
 --startmode=runonce
 --env=SM_CLIENTCONNECT=brokerConnect.conf ic-broker


                   BASEDIR/smarts
               /bin/brstart
 --port=<NEW VALUE>
 --output
 --restore=
                   BASEDIR/smarts
               /local/repos/broker/broker.rps s
```

Disabling the Broker

In the second scenario, you must change the SM_BROKER_DEFAULT host variable in the runcmd_env.sh file and change the **--**startmode value.

1 Change the SM_BROKER_DEFAULT host variable in the runcmd_env.sh file, which is located in the BASEDIR/smarts/local/conf directory:

```
SM_BROKER_DEFAULT=<host>:<value>
```

2 Additionally, you must change the --startmode value in the **sm_service install** Broker command line from **runonce** to **disable**.

Use the **sm_service show** action with the **--**cmdline option to display the existing install Broker command line.

```
                   BASEDIR/smarts
               /bin/sm_service show --cmdline ic-broker



                   BASEDIR/smarts
               /bin/sm_service install
 --startmode=runonce
 --env=SM_CLIENTCONNECT=brokerConnect.conf ic-broker

                   BASEDIR/smarts
               /bin/brstart
 --port=<PORT>
 --output
 --restore=
                   BASEDIR/smarts
               /local/repos/broker/broker.rps
```

```
t
                   BASEDIR/smarts
               /bin/sm_service install --force
  --startmode=disable
  --env=SM_CLIENTCONNECT=brokerConnect.conf ic-broker

                   BASEDIR/smarts
               /bin/brstart
  --port=<PORT>
  --output
  --restore=
                   BASEDIR/smarts
               /local/repos/broker/broker.rps s
```

Securing the Broker

Securing access prevents unauthorized usage of the Broker, and protects it from being modified (for example, Managers being deleted from the Broker). Authenticating users and determining their privileges are accomplished by using two files: serverConnect.conf and brokerConnect.conf. These files ensure that only authorized users access the Broker. You must modify the local copies of these files, which are located in BASEDIR/smarts/local/conf. *"Configuring a secure Broker" on page 108* provides additional information.

Managing Log Files

7

This chapter includes the following topics:

- Overview of message logging
- Retrieving and setting log, error and trace levels at runtime
- Log files
- Starting a new log file
- Controlling the number of saved log files

Overview of message logging

All SDKexecutable processes will create log data, and when the process is started with the **--**output option this data is written to at least one log file. One log file is always created, but there may be two different log files created, depending on the settings of the SM_LOCALE and SM_ENCODING_OUPUT environment variables.

The name of a log file has the following syntax:

```
<name>_
        <locale>_
        <encoding>.log
```

where:

- *<name>* in the log filename column is determined by the rules that apply to the **--**output command option, as described in Command-line options that control VMwaremessages.
- *<locale>*is based on the value of the SM_LOCALE environment variable.
- *<encoding>*is based on the value of the SM_ENCODING_INPUT environment variable . SM_LOCALEand SM_ENCODING_*provide additional information.

  A log file whose locale is en_US (default value of SM_LOCALE) and encoding is UTF-8 (default value of SM_ENCODING_INPUT) is always created when the **--**output option is provided.

- If the SM_LOCALE environment variable specifies the English locale (en_US), and the SM_ENCODING_OUTPUT environment variable specifies UTF-8, then the English log is the only log file produced.
- If the SM_LOCALE variable specifies a locale other than en_US **,** or the SM_ENCODING_OUTPUT environment variable specifies something other than UTF-8, a second log file is created. Log messages written to this second log are localized according to the setting of the SM_LOCALE variable, if such a localization exists.
- If SM_LOCALE specifies an invalid locale (one for which SDKsoftware has not been localized), English log data will result in the process log file.

  Log filenames, locales and encodingssummarizes the filename, locale and encoding of the log files that are produced, including the system log containers.

Table 7-1. Log filenames, locales and encodings

Log use	Log filename (if –output is used)	Locale	Encoding
English log	*<name>*_en_US_UTF-8.log For example, MYDM_en_US_UTF-8.log	en_US	UTF-8
Non-english log	*<name>*_ *<locale>*_ *<encoding>*.log For example, MYDM_ja_JP_<encoding>.log	Defined by SM_LOCALE	Specified by the SM_OUTPUT_ENCODING variable.
UNIX syslog		lDefined by SM_LOCALE	UTF-8 (permitted and expected by the new IETF syslog-protocol RFC, still in a draft form)
Win Event Log		Defined by SM_LOCALE	UTF-16

SM_LOCALE and SM_OUTPUT_ENCODING examplesprovides several examples of SM_LOCALE and SM_ENCODING_OUTPUT variable settings and the resultant logs.

Table 7-2. SM_LOCALE and SM_OUTPUT_ENCODING examples

SM_LOCALE setting	SM_OUTPUT_ENCODING value	Logs that are produced
en_US	UTF-8	*<name>*_en_US_UTF-8.log
en_US	ShiftJIS	*<name>*_en_US_UTF-8.log *<name>*_en_US_ShiftJIS.log
fr	UTF-8	*<name>*_en_US_UTF-8.log *<name>*_fr_UTF-8.log
zh	GB	*<name>*_en_US_UTF-8.log *<name>*_zh_GB.log

The volume of output to the system log containers is controlled by the **--**loglevel option. The volume of output to the log files is controlled by the -–errlevel option, and by the environment variables that control filtering on specific error tables and error codes.

Log file management behavior applies to both log files. Log rolling is managed in a synchronized fashion so that data in multiple logs can be correlated. Thus, when the English log rolls, the non-English log, if it is being produced, will also roll.

Command-line options that control VMware messages

SDKmessages are controlled by the following standard options for the sm_server sm_service command and environment variables.

For the sm_service sm_server options that use a minimum security level, the level can be: none, emergency, alert, fatal, critical, error, warning, notice, informational, and debug. Note that fatal is equivalent to emergency, alert, or critical. Foundation software (DMT) message loggingdisplays the range of security levels.

- Option **--**output[= <*file*>]

  Write all logging output for an SDKapplication to an SDKlog file in the BASEDIR/smarts/local/logs directory. The name of the log file is <*file*>_ <*locale*>_ <*encoding*>.log, or the **--**logname value if <*file*>is omitted. If **--**output is not specified in the invocation line, then no logs are produced.

- Option **--**errlevel= <*min_severity_level*>

  Used by the SDKexception logging mechanism to determine the minimum exception level for writing messages to the log files. By default, **--**errlevel is set to warning. Exceptions below this level are not written to the log files.

- Option **--**loglevel= <*min_severity_level*>

  Used by the SDKexception logging mechanism to determine the minimum exception level for sending messages to the system error logger. By default, **--**loglevel is set to error. Exceptions below this level are not forwarded to the system error logger.

- Option --logname= <*name*>

  Used to specify the name of the SDKlog file and the name of the sending application in messages destined for the system error logger. The default is the name of the application specified by the **--**name option (or just **-**n).

- Option **--**tracelevel= <*min_severity_level*>

  Used to print a stack trace to the SDKlog file when an exception at this level or above occurs. By default, **--**tracelevel is set to fatal. Exceptions below this level do not write a stack trace.

- Environment variable SM_MAIN_OPTIONS

  Used to set the defaults for the standard options for all SDKapplications. For example,

  ```
  SM_MAIN_OPTIONS=--loglevel=None
  ```

- Environment variable SM_LOGFILES

  Used to change the default log location for SDKlog files from BASEDIR/smarts/local/logs to a user-specified location.

Example

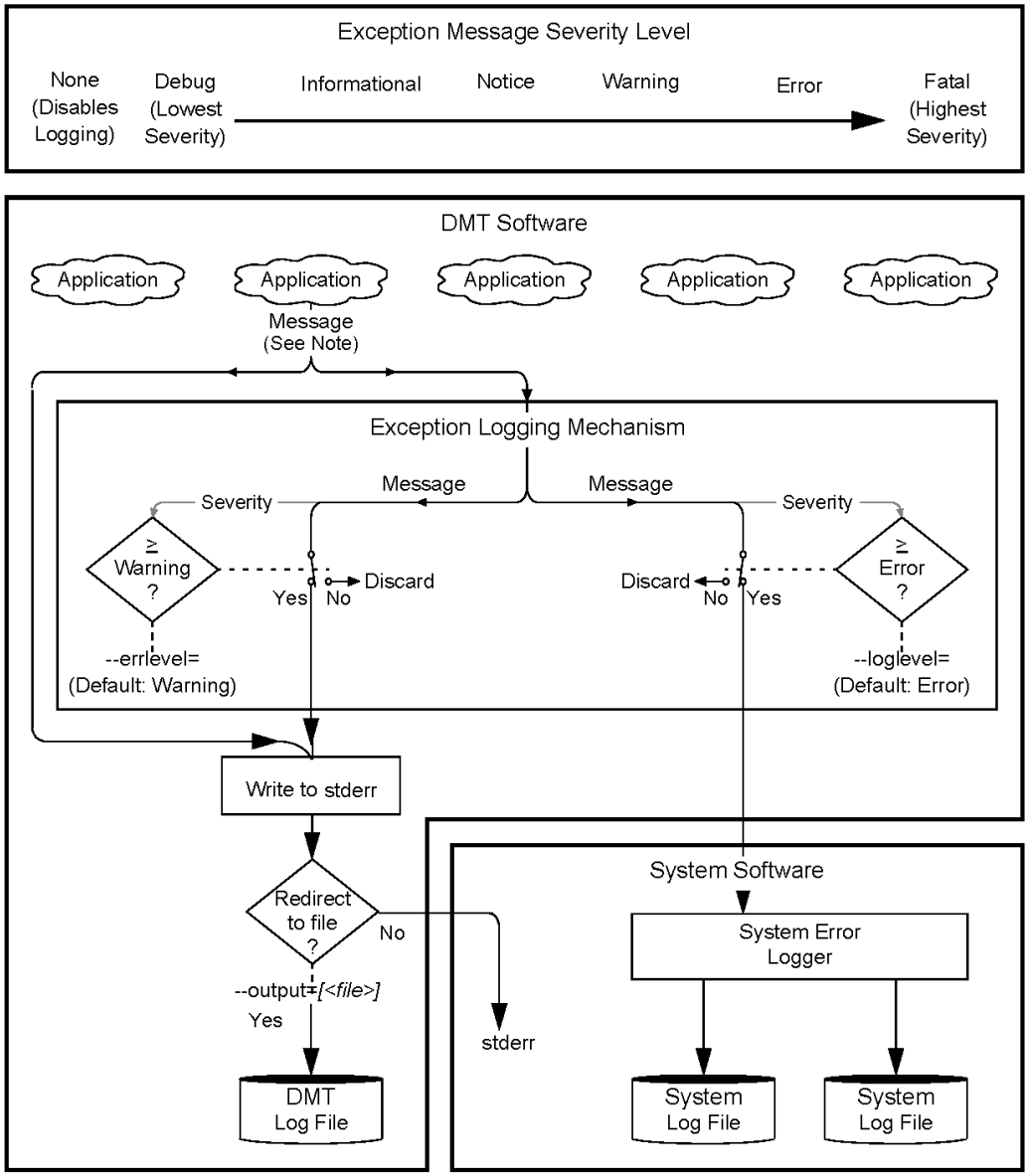
```
SM_LOGFILES=/u/ps/johndoe/ic/logs
```

*The sm_service command line on page 68*provides additional information about sm_service command options. *Starting a Domain Manager on page 60*provides additional information about sm_server command options. Chapter 8 Environment Variables Used by VMware Softwareprovides additional information about environment variables.

Note In Foundation software (DMT) message logging, the levels emergency, alert, and critical are not shown; fatal is equivalent to emergency, alert, or critical.

a

Figure 7-1. Foundation software (DMT) message logging



Note An SDKapplication that makes use of the SDKlogging mechanism will make use of the flow in Foundation software (DMT) message loggingto log message to either the SDKlog file or to the exception logging mechanism. Applications that write directly to stdout or stderr will not abide by this flow diagram.

Retrieving and setting log, error and trace levels at runtime

There are three computed attributes available to get and set the log, error, and trace levels of a Domain Manager at runtime. These computed attributes, described in Computed attributes to retrieve and set log, error, and trace levels at runtime, are available on the SM_JIIM_Support object, and are therefore available from within JIIM code. The SM_System object is a subclass of SM_JIIM_Support and also inherits these computed attributes. The values of these computed attributes can be retrieved and set by using dmctl.

Table 7-3. Computed attributes to retrieve and set log, error, and trace levels at runtime

Computed Attributes	Description
logLevel	The minimum exception level for sending messages to the system error logger.The logLevel attributeis a string, and can be any one of the values set for the --loglevel command line option.
errLevel	The minimum exception level for writing messages to the log files. The errLevel attributeis a string, and can be any one of the values set for the --errlevel command line option.
traceLevel	Used to print a stack trace to the SDK log file when an exception at this level or above occurs. Exceptions below this level do not write a stack trace. The traceLevel attributeis a string, and can be any one of the values set for the --tracelevel command line option.

Retrieving the current level

To retrieve the current levels, invoke the get method on SM_System::SM-System::logLevel, SM_System::SM-System::errLevel, or SM_System::SM-System::traceLevel. Invoking the get method will return a string representing the current level such as "Warning", "Error", or "Fatal".

In this example the error level setting in the MYDM Domain Manager is retrieved, by using dmctl:

dmctl -s MYDM get SM_System::SM-System::errLevel

```
Warning
```

Setting the level

To change the current levels, invoke the put method on SM_System::SM-System::logLevel, SM_System::SM-System::errLevel, or SM_System::SM-System::traceLevel, and the level is changed appropriately.

In this example, the trace level setting is changed to None in the MYDM Domain Manager.

```
dmctl -s MYDM put SM_System::SM-System::traceLevel None
```

When you change the log, error or trace levels by using the put method on one of these computed attributes, dmctl does not return anything. However, a message is printed in the log file. The log message will appear similar to the following

```
[April 8, 2009 5:03:41 PM EDT +122ms] t@1149000000 SM_ProtocolEngine-6
JM_MSG-*-JM_TRACE_LEVEL_CHANGED-User 'user1', using remote dmctl client (id 6), on host host1 with
credentials tpadmin1 has changed the Trace level to None; in file
"/mypath/repos/jiim/SM_JIIM_Support_Impl.c" at line 458
```

Log files

When the --output option is specified for the sm_server sm_service command, each Manager stores its working history in a log file. Every time a Manager starts, it writes information to the log file when one of the following occurs:

- Error
- Connection is lost

Log files are, by default, written to the BASEDIR/smarts/local/logs directory. The SM_WRITEABLE environment variable specifies the default locations for log files and other output files such as repositories and saved consoles, as described in SM_WRITEABLE.

Also, the location where log files are saved can be separately controlled with the SM_LOGFILES environment variable, as described in SM_LOGFILES.

All log files have a .log file type. The default name of a log file is based on the Manager's name. For example, if the name of the Manager is MYDM, then the name of the English log file is MYDM_en_US_UTF-8.log. To change the default name, specify a filename with the --output option for the sm_server sm_service command.

The names of log files written by adapters are described in their respective user’s guides.

SDK programs can maintain up to 1,000 different copies of backup log files for each log that is being generated. For example, if you are generating two log files, MYDM_en_US_UTF-8.log and MYDM_fr_UTF-8.log file, you can maintain up to 1,000 backup log files for the MYDM_en_US_UTF-8.log and 1,000 backup log files for the MYDM_fr_UTF-8.log.

The number of saved log files is determined by the value of the SM_BACKUP_FILE_LIMIT environment variable. When a Manager starts up, it renames a file that matches its log filenames and adds a .bak suffix to the name (for example, MYDM_en_US_UTF-8.log to MYDM_en_US_UTF-8.log.bak). If a file with this name already exists, it is renamed MYDM_en_US_UTF-8.log.NNN, and a new MYDM_en_US_UTF-8.log.bak is created. By default, SDK programs save two log files, which does not include the active log file.

Log file management behavior applies to both log files. Log rolling is managed in a synchronized fashion so that data in multiple logs can be correlated. Thus, when the non-English log rolls, the English log, if it is being produced, will also roll.

Diagnostic header

Every log file (initial and after any log rolling) starts with a diagnostic header which indicates the startup and current information for an SDK process.

The diagnostic header is as follows:

```
MAIN_MSG-*-STARTUP-The following lines are routine diagnostic output; this is not a crash.
================================================================================
Start of process debug and runtime information:
================================================================================
```

<the standard header included with a stacktrace>

```
================================================================================
End of process debug and runtime information
================================================================================
```

The standard header for a stacktrace includes:

- Command line arguments of the process
- Which libraries are loaded
- The environment

Example

```
[May 18, 2009 3:18:37 PM EDT +399ms] t@4139505360 platform
MAIN_MSG-*-STARTUP-The following lines are routine diagnostic output; this is not a crash.
Process startup and runtime information:
================================================================================
Linux 206180000:
IP_NETWORK_SUITE: V10.0.0.0(89403), 14-Apr-2019 16:07:05 Copyright 2019, VMware Inc - Build 106
sm_server --name=Network-Resource --port=46429 --config=icf-r --bootstrap=bootstrap.conf --ignore-
restore-errors --output
DMT: V10.0.0.0(89265), 08-Apr-2019 22:02:59 Copyright 2019, VMware Inc - Build 1161
libsm_clsapi.so (89265) 09-May-2019 02:04:49 UTC
libsm_servif.so (89265) 09-May-2019 02:06:35 UTC
sm_server (89266) 01-Apr-2019 02:58:21 UTC
IP_NETWORK_SUITE: V10.0.0.0(89403), 14-Apr-2019 16:07:05 Copyright 2019, VMware Inc - Build 106
    time=1242674317
    SM_HOME=/opt/InCharge8/IP/smarts
    CWD=/
    Initial uid=0 euid=0 gid=0 egid=0
    Current uid=0 euid=0 gid=0 egid=0
Environment:
    REMOTEHOST=USSAHATHAJL2C.corp.vmware.com
    HOSTNAME=docvm-lx1
    TERM=xterm
    SHELL=/bin/bash
    HISTSIZE=1000
    SM_NEED_PIPEDIR=YES
```

```
    KDE_NO_IPV6=1
    WINDOWID=8388624
    SM_RUNCMD_DEPTH=1
    QTDIR=/usr/lib64/qt-3.3
    QTINC=/usr/lib64/qt-3.3/include
    SM_FIXED=/opt/InCharge8/IP/smarts
    SM_LICENSE=17955@smarts-lmgr92.smarts.com
    XTERM_SHELL=/bin/bash
    USER=root

LD_LIBRARY_PATH=/opt/InCharge8/IP/smarts/local/lib:/tmp/istemp5930135131105/_bundledJRE_/lib/i386/serve
r:/tmp/istemp5930135131105/_bundledJRE_/lib/i386:/tmp/istemp5930135131105/_bundledJRE_/../lib/i386:/op
t/InCharge8/IP/smarts/lib

LS_COLORS=no=00:fi=00:di=00;34:ln=00;36:pi=40;33:so=00;35:bd=40;33;01:cd=40;33;01:or=01;05;37;41:mi=01;
05;37;41:ex=00;32:*.cmd=00;32:*.exe=00;32:*.com=00;32:*.btm=00;32:*.bat=00;32:*.sh=00;32:*.csh=00;32:*.
tar=00;31:*.tgz=00;31:*.arj=00;31:*.taz=00;31:*.lzh=00;31:*.zip=00;31:*.z=00;31:*.Z=00;31:*.gz=00;31:*.
bz2=00;31:*.bz=00;31:*.tz=00;31:*.rpm=00;31:*.cpio=00;31:*.jpg=00;35:*.gif=00;35:*.bmp=00;35:*.xbm=00;3
5:*.xpm=00;35:*.png=00;35:*.tif=00;35:
    SUDO_USER=gossea
    SUDO_UID=53761
    SM_OUTGOING_PROTOCOL=1,0
    KDEDIR=/usr
    POSIXLY_CORRECT=y
    SM_BROKER_DEFAULT=localhost:426
    NLSPATH=/usr/dt/lib/nls/msg/%L/%N.cat
    SM_IP_VERSIONS=v6v4

PATH=/opt/InCharge8/IP/smarts/local/bin/system:/opt/InCharge8/IP/smarts/bin/system:/opt/InCharge8/IP/sm
arts/bin:/work/gossea/bin:/bin:/usr/bin:/opt/gnu/bin:/opt/misc/bin:/bin
    MAIL=/var/spool/mail/gossea
    SM_DEFHOME=/dev/null
    SM_HOME=/opt/InCharge8/IP/smarts
    PWD=/
    INPUTRC=/etc/inputrc
    SM_SITEMOD=/opt/InCharge8/IP/smarts/local:/opt/InCharge8/IP/smarts
    SM_INCOMING_PROTOCOL=1,0
    KDE_IS_PRELINKED=1
    LANG=en_US.UTF-8
    SM_SNMP_BUG_COMPATIBLE=TRUE
    XFILESEARCHPATH=/usr/dt/app-defaults/%L/Dt
    SM_WRITEABLE=/opt/InCharge8/IP/smarts/local
    LM_LICENSE_FILE=17955@smarts-lmgr92.smarts.com
    XTERM_VERSION=XTerm(215)
    SSH_ASKPASS=/usr/libexec/openssh/gnome-ssh-askpass
    SM_PIPEDIR=/var/smarts/pipes/
    HOME=/vmware/gossea
    SHLVL=4
    SUDO_COMMAND=./setup-IP-10_0_0_0-linux.bin
    SM_DEF=/dev/null

SM_POP_1=PATH=/opt/InCharge8/IP/smarts/bin:/work/gossea/bin:/bin:/usr/bin:/opt/gnu/bin:/opt/misc/bin:/b
in
    SM_POP_3=LM_LICENSE_FILE=
    REMOTEUSER=gossea
```

```
    LOGNAME=root

SM_POP_2=LD_LIBRARY_PATH=/tmp/istemp5930135131105/_bundledJRE_/lib/i386/server:/tmp/istemp593013513110
5/_bundledJRE_/lib/i386:/tmp/istemp5930135131105/_bundledJRE_/../lib/i386
    CVS_RSH=ssh
    QTLIB=/usr/lib64/qt-3.3/lib
    SM_POP_5=SM_BROKER=
    SM_PROCESS_PATH=/opt/InCharge8/IP/smarts/bin/system/sm_server
    SM_POP_4=SM_LIBPATH=
    SM_POP_7=SM_PROCESS_PATH=
    LESSOPEN=|/usr/bin/lesspipe.sh %s
    SM_POP_6=SM_BROKER_DEFAULT=
    SM_BROKER=localhost:426
    SM_LOCALE=
    DISPLAY=10.4.145.237:0.0
    SUDO_GID=201

SM_LIBPATH=/opt/InCharge8/IP/smarts/local/lib:/tmp/istemp5930135131105/_bundledJRE_/lib/i386/server:/tm
p/istemp5930135131105/_bundledJRE_/lib/i386:/tmp/istemp5930135131105/_bundledJRE_/../lib/i386:/opt/InCh
arge8/IP/smarts/lib
    G_BROKEN_FILENAMES=1
    CMP_OSLS_PROCESS_PATH=/opt/InCharge8/IP/smarts/bin/system/sm_server/sm_server
Current limit on CORE file size: 4294967295
Current top of heap: 167768064
================================================================================
[May 18, 2009 3:19:53 PM EDT +254ms] t@4052745104 InCharge Framework
ICF_MSG-*-ICF_LICENSEDFEATURE-Enabling the following licensed feature(s):
      InCharge Availability Analysis
      InCharge Performance Analysis
      InCharge IPv6
      Certification of All Vendor Devices
      Certification for Cisco Devices
      Certification for Cisco HSRP support
      Incharge Resource Manager
      InCharge Autodiscovery
      InCharge NAS Host Discovery
      InCharge Virtual Router Discovery
      InCharge Host Resources Discovery
      Alcatel 5620 SAM User
[May 18, 2009 3:19:53 PM EDT +264ms] t@4052745104 InCharge Framework
ICF_MSG-*-ICF_RESTORESTART-PersistenceManager: restore started
[May 18, 2009 3:19:55 PM EDT +349ms] t@4052745104 InCharge Framework
MR-W-RPS_FORMAT_MISMATCH-Format of value for
    'ICF_TopologyManager::ICF-TopologyManager::FieldSeparator' differs from
    the expected type 'Byte'
[May 18, 2009 3:19:55 PM EDT +350ms] t@4052745104 InCharge Framework
MR-W-RPS_FORMAT_ERROR-Error in format value specified for
    'ICF_TopologyManager::ICF-TopologyManager'; in file
    "/work/redcurrent/DMT-10.0.0.0/1161/smarts/repos/mr/dyn_acc.c" at line 3300
 <OK>-<OK>-Normal successful completion
.
.
.
```

Date and timestamp, and thread id in each log message

The date and timestamp is written at the beginning of each log record. The thread id is also included in each log record, following the date and timestamp. For example:

```
[March 2, 2009 3:31:18 PM EDT +357ms] t@3012557744 SM_ProtocolEngine-1
SVIF_MSG-*-DSM-SM_DebugServer: purgeObserver()
```

Message at the end of a rolled log

When rolling a log, a message is printed at the bottom of the log file before it is rolled. The message is similar to the following example:

```
MAIN-N-Closing this log file at March 19, 2009 9:48:19 PM EDT; continuing in
<BASEDIR>/local/logs/MYDM_en_US_UTF-8.log
```

Starting a new log file

You can request that a Manager or adapter create a new log file, often referred to as *rolling over* a log file. Typically, this is done when a log file becomes quite large. Log files grow indefinitely, though slowly, under normal conditions.

Most log files are small and rolling over a log file is not necessary. However, if you need to start a new log file, use the following methods:

- For UNIX systems only, use the kill or roll_log commands.

  You can repeat the log file roll-over process as many times as you like. If you need to retain log files beyond the limit specified by the SM_BACKUP_FILE_LIMIT environment variable, copy or at least rename the .log.bak files as soon as they are created.

  Log rolling is managed in a synchronized fashion so that data in multiple logs can be correlated. Thus, when the English log rolls, the non-English log, if it is being produced, will also roll.

The kill command method for UNIX systems

On UNIX systems only, it is possible to request that a Manager or an adapter process roll over its log files. You do this by sending a SIGUSR1 signal to the process ID writing the log file. Use the kill command of your shell:

```
kill -USR1 <pid>
```

Upon receipt of the USR1 signal, a Manager or adapter process first writes an informational message to the end of the current log file, repeats the steps it executed at startup, and creates a new log file and a backup log file. All subsequent logging information is recorded to the new file.

Note To determine the process number <*pid*>, you can use the brcontrol utility or for an adapter, use the system™s ps command.

The roll_log command method

The roll_log command is invoked through the dmctl utility, which requires that you attach to a Manager or adapter with administrative privileges. The syntax of the command is:

```
exec roll_log [<file-name>]
```

The <*file-name*> option enables you to specify a new name for current and subsequent log files. If you omit this option, the current naming convention is retained. If you specify a name, the new log file uses that name. Any new log files created with roll_log will also use this name if a different name is not specified. The new name specified by <*file-name*> is handled in exactly the same manner as the --output option of the sm_server command.

For example, on UNIX systems, the equivalent command is:

```
   BASEDIR/smarts
/bin/dmctl -s MYDM exec roll_log
```

When roll_log is invoked, the Manager or adapter does the following:

1. Writes an informational message to the end of the current log files
2. Repeats the steps it executed at startup
3. Creates a new log file and a backup log file for each log file that was currently running.

   All subsequent logging information is recorded to the new files.

Controlling the number of saved log files

SDK software can retain up to 1,000 rolled over log files for each log that is being generated. For example, if you are generating two log files, MYDM_en_US_UTF-8.log and MYDM_fr_UTF-8.log file, you can maintain up to 1,000 backup log files for the MYDM_en_US_UTF-8.log and 1,000 backup log files for the MYDM_fr_UTF-8.log.

The number of log files retained is determined by the value of SM_BACKUP_FILE_LIMIT. When a Manager starts up, it renames a file that matches its log filename, adding a .bak to the name. If there is an existing .bak file, the .bak will be replaced with .NNN.

For example, if the SM_BACKUP_FILE_LIMIT environment variable is set to 3, the process is as follows:

1. A log file is created and it is named MYDM_en_US_UTF-8.log.

2 The MYDM_en_US_UTF-8.log file created in step 1 is rolled over:
   - The MYDM_en_US_UTF-8.log file created in step 1 is renamed to MYDM_en_US_UTF-8.log.bak.
     - A new MYDM_en_US_UTF-8.log file is created.
3 The MYDM_en_US_UTF-8.log file created in step 2 is rolled over:
   - The MYDM_en_US_UTF-8.log.bak file renamed in step 2 is now renamed to MYDM_en_US_UTF-8.log.001.
     - The MYDM_en_US_UTF-8.log file created in step 2 is renamed to MYDM_en_US_UTF-8.log.bak.
     - A new MYDM_en_US_UTF-8.log file is created.
4 The MYDM_en_US_UTF-8.log file created in step 3 is rolled over:
   - The MYDM_en_US_UTF-8.log.bak file renamed in step 3 is renamed to MYDM_en_US_UTF-8.log.002.
     - The MYDM_en_US_UTF-8.log file created in step 3 is renamed to MYDM_en_US_UTF-8.log.bak.
     - A new MYDM_en_US_UTF-8.log file is created.
5 The MYDM_en_US_UTF-8.log file created in step 4 is rolled over:
   - The MYDM_en_US_UTF-8.log.bak file renamed in step 3 is renamed to MYDM_en_US_UTF-8.log.003.
     - The MYDM_en_US_UTF-8.log file created in step 4 is renamed to MYDM_en_US_UTF-8.log.bak.
     - A new MYDM_en_US_UTF-8.log file is created.
6 Since the limit of MYDM_en_US_UTF-8.log.NNN files that are retained is set to 3, when the MYDM_en_US_UTF-8.log file created in step 5 is rolled over:
   - The oldest log file, MYDM_en_US_UTF-8.log.001 from step 3 , is deleted.
     - The MYDM_en_US_UTF-8.log.bak file renamed in step 5 is renamed MYDM_en_US_UTF-8.log.004.
     - The MYDM_en_US_UTF-8.log file created in step 5 is renamed MYDM_en_US_UTF-8.log.bak.
     - A new MYDM_en_US_UTF-8.log is created.

Environment Variables Used by VMware Software

8

This chapter includes the following topics:

- How variable values are interpreted
- Methods for setting environment variables for VMware software
- Environment variables for users

How variable values are interpreted

This section describes how SDK software interprets the value of an environment variable. If a variable, such as SM_BROKER_DEFAULT, requires that its value be formatted in a particular manner, it is noted in the description of the variable.

How integer variables are interpreted

When the value of an environment variable is to be expressed as an integer, SDK software interprets the value of the variable as follows:

- Any leading white space is skipped. If the next character is a plus sign (+) or a minus sign (-), it is also skipped. If the next character was a minus sign, the final numeric value is negated.
- If the value starts with a number other than zero, it may contain only the digits zero through nine, and it is considered a decimal number.
- If the value starts with “0x” or “0X”, it may contain the digits zero through nine, as well as the letters “a” through “f” and “A” through “F”. At least one digit or letter must follow the “x” or “X”. The value is treated as a hexadecimal number.
- If the value starts with a zero and is not followed by a “x” or “X”, it may contain only the digits zero through seven, and it is treated as an octal number.

  Otherwise, the variable does not represent a numeric value. In most cases, an error message is printed. In a few cases, the numeric value is simply taken to be zero.

  Examples of numeric values provides examples of numeric values and how they are interpreted by SDK software.

Table 8-1. Examples of numeric values

Numeric value	Meaning
"0"	The value zero.
" 10"	The value ten. The leading white space is ignored.
"+010"	The value eight, octal.
"0x10"	The value sixteen, hexadecimal.
"-123"	The value negative one hundred twenty-three.
"abc"	An illegal value, may be treated as zero.
"019"	An illegal value, leading zero makes it octal, but octal numbers cannot contain a nine. This value may be treated as zero.
"1 "	An illegal value, leading white space is ignored but trailing white space is not. This value may be treated as zero.
"0X"	An illegal value, at least one digit or letter must follow the “OX”. This value may be treated as zero.
"- 23"	An illegal value, embedded white space is not permitted. This value may be treated as zero.

How Boolean variables are interpreted

When an environment variable is expressed as a Boolean, SDK software interprets the value of the variable, when set, as follows:

- If the value starts with a letter, an uppercase “T” or a lowercase “t” or an uppercase “Y” or a lowercase “y”, the variable is interpreted as true. If the value starts with any other letter, the variable is interpreted as false.
- If the value can be interpreted as an integer variable, and it has a value of zero, the variable is interpreted as false. If the value is anything other than zero, the variable is interpreted as true.
- For any other value, the variable is interpreted as false.

Methods for setting environment variables for VMware software

You can use one of the following methods to set an environment variable. The difference between these methods is that one sets the environment variable across an SDK product suite and the other sets the environment variable for a particular program.

Setting an VMware environment variable suite-wide

When necessary, you can set an environment variable so that an SDK product suite installed in that location can use it. For example, the SM_BROKER_DEFAULT variable, the value of which is set during installation, is used in this manner.

To set an environment variable so that it can be used by the programs of an SDK product suite, add it to the runcmd_env.sh file, which is located in the BASEDIR/smarts/local/conf directory of that product suite.

1 Use sm_edit to open the runcmd_env.sh file. Invoke sm_edit from the *BASEDIR/smarts* /bin directory:

```
sm_edit conf/runcmd_env.sh
```

2 Add the environment variable and its value by using the following syntax:

```
SM_AUTHORITY="<STD>"
```

3 Save the runcmd_env.sh file and close it.

4 Any SDK program within a product suite started after this point will use the applicable environment variables specified in the runcmd_env.sh file. SDK programs that are already running need to be restarted for any new environment variable to take effect.

Setting an VMware environment variable for a program

You can also set an environment variable so that it only affects a particular program. Typically, this is done by adding the environment variable to the program™s sm_service command line. For example, if you want a Manager to use a particular serverConnect.conf file, you would specify the SM_SERVERCONNECT variable in the sm_service command line for that program.

The following example sets the value of the SM_SERVERCONNECT environment variable to point to the server_Connect_IP.conf file.

1 Use the **sm_service show** action with the **--**cmdline option to display the existing command line for the program. You must have administrative privileges when typing this command.

```
t
                 BASEDIR/smarts
              /bin/sm_service show --cmdline ic-am-server s


                 BASEDIR/smarts
              /bin/sm_service install
--startmode=runonce ic-am-server

                 BASEDIR/smarts
              /bin/sm_server
  --name=INCHARGE-AM
  --config=icf
   --bootstrap=bootstrap-am.conf
   --port=0
   --ignore-restore-errors
   --output
```

2 Use the **sm_service install** action with the **--**force option to add the environment variable to the command line. The **--**env option specifies the environment variable.

```
t
                  BASEDIR/smarts
              /bin/sm_service install --force
--startmode=runonce
--env=SM_SERVERCONNECT=
                  BASEDIR/smarts
              /conf
/serverConnect_IP.conf
ic-am-server

                  BASEDIR/smarts
              /bin/sm_server
  --name=INCHARGE-AM
  --config=icf
  --bootstrap=bootstrap-am.conf
  --port=0
  --ignore-restore-errors
   --output s
```

3 Do one of the following:

- Start the program.
- Stop and restart the program.

Environment variables for users

The following list of environment variables can be applied to SDK software.

- SM_DISPLAY
- SM_EDITOR
- SM_SNMP_BUG_COMPATIBLE
- SM_SNMP_MAX_MESSAGE_SIZE
- SM_ENABLE_SNMP_SET
- SM_MAIN_OPTIONS
- SM_OUTGOING_PROTOCOL
- SM_INCOMING_PROTOCOL
- SM_DISABLE_KEEPALIVES
- SM_IP_VERSIONS
- SM_MAX_MONITORING_THREADS
- SM_AUDIT_CLIENT_ACTIONS

- SM_AUDIT_ALL_CONNECTIONS
- SM_NO_LINE_FOLD
- SM_FOLD_WIDTH
- SM_HOSTSFILE
- SM_LOCALE
- SM_ENCODING_SHELL
- SM_ENCODING_SYSTEM
- SM_ENCODING_INPUT
- SM_ENCODING_OUTPUT
- SM_JAVA_PACKAGE_ROOT
- SM_JDK_HOME
- SM_JAVA
- SM_JAVAHOME
- SM_BROKER_DEFAULT
- SM_BROKER
- LM_LICENSE_FILE
- SM_LICENSE
- SM_LMGRD_LICENSE_FILE
- SM_AUTHORITY
- SM_KEYFILE
- SM_CLIENTCONNECT
- SM_OKLOGIN_LEVEL
- SM_SERVERCONNECT
- SM_DATETIME_FORMAT
- TZ
- LC_TIME
- SM_BACKUP_FILE_LIMIT
- SM_WRITEABLE
- SM_LOGFILES
- SM_RULESET_PATH
- SM_COUNT_REPOS_LOCKS
- SM_REPOS_LOCKS_LIMIT

SM_DISPLAY

This environment variable specifies the location of a client's X Window System display (typically for a console user). Output from a server tool that uses the X protocol is directed to the display specified by SM_DISPLAY. Only VMware Smart Assurance Service Assurance Manager uses this variable.

SM_EDITOR

This environment variables enables you to set the text editor that is invoked by the sm_edit utility. When SM_EDITOR is not defined, the sm_edit utility uses the value of one of the following system environment variables: VISUAL then EDITOR. If these environment variables are not defined, the sm_edit utility uses the following editors:

- Edit editor on UNIX systems

SM_SNMP_BUG_COMPATIBLE

SDK implementation of the SNMP protocol is strict in that SDK software only sends or receives SNMP messages that conform to the SNMP standard. By default, SDK software only accepts conforming SNMP messages. Unfortunately, not all SNMP implementations are as strict. The result is that SDK programs may receive nonconforming SNMP messages.

In some cases, SDK software can successfully interpret and handle nonconforming messages. If you set the SM_SNMP_BUG_COMPATIBLE environment variable, SDK software will accept the nonconforming SNMP messages in two cases:

- When the type of an SNMP variable is not the required type but the value can be unambiguously interpreted as the required type. An example is when a TimeTicks variable type is sent as an Unsigned32 or non-negative Integer32 variable.
- When the type of an SNMP variable is inconsistent with the version of the SNMP protocol that delivers the message. An example is when a Counter64 variable is sent in a version 1 (v1) SNMP message.

  When the SM_SNMP_BUG_COMPATIBLE environment variable is set, any invalid SNMP message that is accepted by the SDK software is logged at a warning level to pinpoint the source of the invalid SNMP message and to determine the reason why it is invalid.

  SM_SNMP_BUG_COMPATIBLE is a Boolean.

SM_SNMP_MAX_MESSAGE_SIZE

SM_SNMP_MAX_MESSAGE_SIZE is an environment variable that controls the maximum size of the SNMP datagram that is processed. The default value is 8k (8192 bytes).

The default size can be overridden by setting the environment variable SM_SNMP_MAX_MESSAGE_SIZE to the required maximum size. For example, setting SM_SNMP_MAX_MESSAGE_SIZE equal to 16384 sets the maximum to 16k, which is double the default value.

If you try to set SM_SNMP_MAX_MESSAGE_SIZE to less than 1472 (the recommended minimum for UDP/IPv4 per RFC-3417, section 3.2), it will be silently adjusted to be exactly 1472.

Note Although SM_SNMP_MAX_MESSAGE_SIZE can be used to set the maximum SNMP datagram size to less than the default 8k, this is *strongly* discouraged.

If you try to set SM_SNMP_MAX_MESSAGE_SIZE greater than 64k (65536 bytes), it will be set to exactly 65536. A value of 65536 bytes is the largest possible datagram size supported by UDP.

Important Setting SM_SNMP_MAX_MESSAGE_SIZE to large values will increase server memory usage. This is especially true for trap processing during periods of high trap reception rate. Therefore, this option should only be used when it is needed to communicate with SNMP devices sending unusually large datagrams. Only increase the setting of this environment variable on the specific Domain Managers which manage those devices.

SM_ENABLE_SNMP_SET

This environment variable is disabled by default.

- When set to 1, it enables the use of the SM_SNMP_Actions::Set( ) method.
- When it is disabled (set to 0), the method cannot be used to set the value of an OID.

SM_MAIN_OPTIONS

The environment variable SM_MAIN_OPTIONS enables you to set default values for the standard options supported by SDK software, as described in Standard options supported by SM_MAIN_OPTIONS. You can override the values specified by SM_MAIN_OPTIONS by providing alternative values on the command line.

Table 8-2. Standard options supported by SM_MAIN_OPTIONS

Standard option	Definition
--accept=<*hosts*>	Specify a list of hosts, by name or IP address, from which SDK programs accept connections. Multiple entries should be separated by commas. By default, SDK programs accept connections from any host. SDK programs that accept connections are dmstart, sm_server, sm_adapter, and sm_trapd. Note that sm_adapter and sm_trapd accept only connections when started with the --name option. To limit connections to the host on which SDK software is installed, specify the name or IP address of the host instead of localhost.
--daemon	Run the SDK program as a daemon (UNIX only). Messages generated from internal logging calls are directed to the log file. Log files are written to the BASEDIR/smarts/logs directory.

Table 8-2. Standard options supported by SM_MAIN_OPTIONS (Continued)

Standard option	Definition
--facility=<*facility*>	Instructs the SDK program to log any syslog messages with the specified syslog facility (UNIX/Linux only). The default is Daemon. The <*facility*> setting can be one of the following: ■ Kern ■ User ■ Mail ■ Daemon ■ Auth ■ Lpr ■ News ■ Uucp ■ Cron ■ Local0 ■ Local1 ■ Local2 ■ Local3 ■ Local4 ■ Local5 ■ Local6 ■ Local7
--errlevel=<*level*>	Specify the minimum level at which error events are written to the standard error output. Possible values include: none, emergency, alert, fatal, critical, error, warning, notice, informational, debug. The default is warning. (Note that fatal is equivalent to emergency, alert, or critical.)
--loglevel=<*level*>	Specify the minimum level at which event messages are written to the system logging facility. Possible values include: none, emergency, alert, fatal, critical, error, warning, notice, informational, debug. The default is error. (Note that fatal is equivalent to emergency, alert, or critical.)
--logname=<*name*>	Specify the name used to identify the SDK program in the system log. On UNIX systems, the default is the program name.
--output[=<*file*>]	Messages generated from internal logging calls are directed to a log file. Log files are written to the BASEDIR/smarts/logs directory. If <*file*> is not specified, the value of --logname is used. If a log file with specified name already exists, it is moved to a backup file, BASEDIR/smarts/local/logs/<*file*>.bak. If the backup file already exists, it is deleted.
--tracelevel=<*level*>	Specify the minimum level at which a process stack trace is written to the standard error output. Possible values include: none, emergency, alert, fatal, critical, error, warning, notice, informational, debug. The default is fatal. (Note that fatal is equivalent to emergency, alert, or critical.)

Table 8-2. Standard options supported by SM_MAIN_OPTIONS (Continued)

Standard option	Definition
--useif=<*IP address*>	Specifies the IP address that the SDK program should use as the source address for outgoing packets and the destination address for which it accepts incoming packets. <*IP Address*> must be a *v4* address.
--version	Print the version of the SDK program.

For example, if you set SM_MAIN_OPTIONS to --errlevel=fatal, then SDK programs will record all errors with a level of fatal or higher.

SM_OUTGOING_PROTOCOL

This environment variable controls the kinds of connections that an SDK program acting as a client is allowed to initiate. It can contain one or more numbers. These numbers specify the security level at which communications can occur. If more than one is specified, they must be separated with commas. If the variable is not set, it is the same as specifying "0".

SM_INCOMING_PROTOCOL

This environment variable controls the kinds of connections that an SDK program acting as a server is allowed to accept. It can contain one or more numbers. If more than one is specified, they must be separated with commas. If the variable is not set, it is the same as specifying "0".

SM_DISABLE_KEEPALIVES

This environment variable controls KeepAlive messages for both Managers and clients. To disable KeepAlive messages, specify a nonzero number. To enable the messages, specify a zero (0).

SDK software uses KeepAlive messages to test TCP connections between SDK components. If a KeepAlive message is not received, a message is written to the Manager's log file. These logged messages may indicate that failed KeepAlive messages may be the result of other software applications and hardware not supporting TCP urgent data, and not the result of the connection being down.

To control KeepAlive messages for only the Global Console, specify the Java property com.smarts.disable_keepalives as described in the VMware Smart Assurance Service Assurance Manager Operator Guide.

SM_IP_VERSIONS

The SM_IP_VERSIONS environment variable enables you to control the address family used for hostname resolution. It affects the following:

- VMware Smart Assurance utilities that use a command line (for example, sm_snmpwalk, dmctl, sm_tpmgr)
- Name resolution during discovery
- ASL scripts
- Ping

- DNS lookup

  SM_IP_VERSIONS also controls which ports a Domain Manager listens on:

- If SM_IP_VERSIONS is set to "v4" or "v6", a Domain Manager will only open sockets and listen on that address family.
- If SM_IP_VERSIONS is set to "v4v6" or "v6v4", a Domain Manager will listen on both address families.

  The SM_IP_VERSIONS setting is also dependent on the address family as follows:

- If you are running on a system which only supports a single address family, it is recommended that SM_IP_VERSIONS be set to "v4" or "v6", as appropriate.
- If you are running on a system which supports both address families, set it to the value which best describes your administrative preferences.

  The variable can be set depending on the order in which you want to do hostname resolution.If the variable is not set, and the address family is not explicitly provided on the command line (for example, frame.someDomain.vmware.com:v4), the default behavior is to resolve the hostname as an IPv4 hostname. The variable should be set to the address family that is predominate for the network.

  To set this variable, add it to the runcmd_env.sh file, which is located in the BASEDIR/smarts/local/conf directory of the product suite.

  The syntax of the environment variable is:

```
SM_IP_VERSIONS="<ip_value>"
```

  Acceptable values for the SM_IP_VERSIONS environment variable lists acceptable values for the SM_IP_VERSIONS environment variable.

Table 8-3. Acceptable values for the SM_IP_VERSIONS environment variable

ip_value	Description
"V4"	The hostname is resolved to an IPv4 hostname (default).
"V6"	The hostname is resolved to an IPv6 hostname.
"V4V6"	The hostname is resolved to an IPv4 hostname. If that fails, the Domain Name System server tries to resolve the hostname to an IPv6 hostname.
"V6V4"	The hostname is resolved to an IPv6 hostname. If that fails, the Domain Name System server tries to resolve the hostname to an IPv4 hostname.
Note: The acceptable value can also be lowercase ("v4", "v6", "v4v6" or "v6v4").	

Detailed instructions about setting environment variables and information about the runcmd_env.sh file is provided in the Methods for setting environment variables for VMware software.

SM_MAX_MONITORING_THREADS

TheSM_MAX_MONITORING_THREADS environment variable represents the maximum number of monitoring threads that the monitoring system will use for monitoring changes. The default value is 3.

The recommended values are described in Recommended values for SM_MAX_MONITORING_THREADS.

Table 8-4. Recommended values for SM_MAX_MONITORING_THREADS

Platform	Recommended value
All supported platforms	Up to 3, but no more than the number of CPUs on the machine.

SM_AUDIT_CLIENT_ACTIONS

Setting the SM_AUDIT_CLIENT_ACTIONS environment variable equal to 1 produces the log if the client requests it. However, even if the client requests it, auditing does not occur if it has not been enabled on the server with either the SM_AUDIT_CLIENT_ACTIONS variable, or the SM_AUDIT_ALL_CONNECTIONS environment variable, which is described in SM_AUDIT_ALL_CONNECTIONS.

SM_AUDIT_ALL_CONNECTIONS

Setting the SM_AUDIT_ALL_CONNECTIONS environment variable equal to 1 forces the logging even if the client does not request it. If you set SM_AUDIT_ALL_CONNECTIONS equal to 1, then you do not have to set SM_AUDIT_CLIENT_ACTIONS, described in SM_AUDIT_CLIENT_ACTIONS.

If you set SM_AUDIT_ALL_CONNECTIONS, you do not need to modify the client. However, if you suspect a specific client you might still want to use the by-request-only access provided by SM_AUDIT_CLIENT_ACTIONS to reduce the amount of logging coming from other sources.

SM_NO_LINE_FOLD

The environment variable SM_NO_LINE_FOLD can be set to prevent the line wrapping in log files. Add the following lines to the BASEDIR/local/conf/runcmd_env.sh file:

```
export SM_NO_LINE_FOLD
SM_NO_LINE_FOLD="TRUE"
```

You can use any string value for SM_NO_LINE_FOLD, it is only if it has no string associated with it that lines fold.

SM_FOLD_WIDTH

The SM_FOLD_WIDTH environment variable removes the default line wrapping of exception messages in the log file. The lines will only wrap if SM_FOLD_WIDTH=X where X specifies the column at which to fold the lines.

SM_HOSTSFILE

On 32-bit Solaris builds of SDK software, a bug in the Solaris system libraries prevents /etc/hosts from being read properly if more than 256 file descriptors are in use. If you set SM_HOSTSFILE=1 in the environment, then /etc/hosts is read using application code, bypassing the system libraries as a work-around to this problem.

Note This work-around only handles IPv4 addresses in /etc/hosts. The work-around does not work for IPv6 addresses, and will not read /etc/ipnodes (or /etc/inet/ipnodes). IPv6 addresses obtained from DNS are handled properly in all cases. If it is necessary to obtain IPv6 addresses from /etc/ipnodes, you should use the 64-bit version of the product.

SM_HOSTSFILE is only available on Solaris versions of SDK software. It operates in both 32 and 64-bit builds, but since the 64-bit system libraries work properly, it is recommended that it *not* be set on 64 bit builds.

Variables that control the locale and encoding

These variables specify the local and encoding used by SDK programs.

SM_LOCALE

The SM_LOCALE environment variable specifies the appropriate language and region in which you want to see output. In general, SM_LOCALE must be specified consistently with advice in IETF RFC 1766/3066. If you do not specify a value for SM_LOCALE, it defaults to en_US.

ICU allows for some flexibility on the case (upper and lower) of the language and country codes and for the separator character used to join the two.

SDK software will convert the value specified in SM_LOCALE to a normalized form that can be recognized by all ICU library functions which take a locale argument. The normalization (done by calling the ICU function, uloc_canonicalize) enables ICU library functions to perform predictably.

```
export SM_LOCALE=ja_JP
sm_server --name=Tokyo
```

SM_ENCODING_*

There are four SM_ENCODING_* environmental variables that can be set.

- SM_ENCODING_SHELL indicates the encoding used by the shell, primarily useful for specifying what encoding is used for command line arguments.
- SM_ENCODING_SYSTEM indicates the encoding used for internal interaction with the operating system.
  - Primarily dictates the encoding used for file system access.
  - Secondarily indicates the encoding used when internally accessing the environment.

- SM_ENCODING_INPUT indicates the encoding used by stdin.
- SM_ENCODING_OUTPUT indicates the encoding used by stdout and stderr.

  Tuning the SM_ENCODING_INPUT and SM_ENCODING_OUTPUT values is primarily useful if you are redirecting or piping text somewhere other than the shell.

  On UNIX the behavior is:

- If SM_ENCODING_SHELL is not set, then its value will default to "UTF-8."
- If SM_ENCODING_SYSTEM is not set, then its value will mirror SM_ENCODING_SHELL.
- If SM_ENCODING_INPUT is not set, then its value will mirror SM_ENCODING_SHELL.
- If SM_ENCODING_OUTPUT is not set, then its value will mirror SM_ENCODING_INPUT.

Note In UNIX it generally makes sense to set all four values identically.

The following example sets the encoding to Shift_JIS.

```
export SM_ENCODING_INPUT=Shift_JIS
export SM_ENCODING_OUTPUT=Shift_JIS
export SM_ENCODING_SHELL=SHIFT_JIS
export SM_ENCODING_SYSTEM=Shift_JIS
```

Variables that control Java settings

These variables specify the Java settings.

SM_JAVA_PACKAGE_ROOT

The SM_JAVA_PACKAGE_ROOT environment variable can be used to specify the java package root.

SM_JDK_HOME

The SM_JDK_HOME environment variable is used to indicate the location of the JDK, if present. If you provide the location of the JDK while running the appropriate setup script, then the SM_JDK_HOME variable will be defined in the runcmd script.

Note If you are using a 64-bit version of the SDK software, then you should set SM_JDK_HOME to the location of a 64-bit version of the JDK. Likewise, if you are using a 32-bit version of the SDK software, then you should set SM_JDK_HOME to indicate the location of a 32-bit version of the JDK.

SM_JAVA

The SM_JAVA environment variable specifies a Java Runtime Environment (JRE) other than that in BASEDIR/smarts/jre. The contents are the paths to the libraries of the JRE. On UNIX, the value of SM_JAVA is added to the LD_LIBRARY_PATH (Solaris and Linux).

This variable is optional. If left unset, runcmd will assume standard paths relative to the value of SM_JAVAHOME.

SM_JAVAHOME

The SM_JAVAHOME environment variable is used in conjunction with SM_JAVA to specify a non-standard JRE. It specifies the location of the JRE, and defines the value of the JVM's "java.home" system property.

- If SM_JAVAHOME is left unset and SM_JDK_HOME was specified, then SM_JAVAHOME will effectively be set to SM_JDK_HOME/jre.
- If left unset and SM_JDK_HOME is also unset, and if the BASEDIR/jre directory exists, then SM_JAVAHOME will effectively be set to BASEDIR/jre.

Variables that control the Broker

These variables control the location of the Broker.

SM_BROKER_DEFAULT

This environment variable specifies the default location of the Broker. When you install SDK software, you are prompted for the host location of the Broker and its port number. The Broker's location can be specified in any one of the following formats: "host:port", "host" or ":port". When the host or port values are omitted, the default value is used. The default host value is "smarts-broker" or "localhost". The default port value is 426. If the location of the Broker changes, you must edit the runcmd_env.sh file on all systems where SDK software is installed. In addition, you may have to edit the Broker startup script.

SDK software follows these steps to determine the Broker's location:

1. Verifies whether the Broker's location was specified as an option at startup. When this option is specified, no other options are checked.
2. It checks the value of the SM_BROKER_DEFAULT environment variable. If this variable is set, no other options are checked.
3. It checks if the Broker is running on the host smarts-broker and listening on TCP port 426.
4. If smarts-broker is not defined, the program checks port 426 on localhost.

Note The hostname smarts-broker is usually an alias, such as a DNS CNAME.

SM_BROKER

SM_BROKER should only be set in special cases when you want to override the default Broker. For example, if you have to temporarily move your Broker in an emergency or test situation, you could set this variable.

Under normal circumstances, you should use the SM_BROKER_DEFAULT environment variable to set the default location of the Broker.

Variables that control VMware security

These variables specify the security used by SDK programs.

SM_AUTHORITY

This environment variable determines the selection of the authority programs used by the SDK software for different parts of the authentication process. For example:

```
SM_AUTHORITY=IDENTIFY=<program1>,AUTHENTICATE=<program2>
```

where:

- *<program1>* is the program granting credentials
- *<program2>* is the program validating credentials.

  Note Program1 and program2 may refer to the same program.

- SM_AUTHORITY=<STD> means:

  ```
  SM_AUTHORITY=IDENTIFY=sm_authority,AUTHENTICATE=sm_authority
  ```

- SM_AUTHORITY=<NONE> means:

  ```
  SM_AUTHORITY=IDENTIFY=sm_authnone,AUTHENTICATE=sm_authnone
  ```

- SM_AUTHORITY=someAuthority means:

  ```
  SM_AUTHORITY=IDENTIFY=someAuthority,AUTHENTICATE=someAuthority
  ```

  Optional arguments can be specified after the authority name. A comma may not be included within the program name. A comma may appear in the arguments if the argument is quoted.

  If SM_AUTHORITY is not specified, SM_AUTHORITY=<STD> is assumed.

SM_KEYFILE

This environment variable specifies the location of the imk.dat file. By default, this variable refers to BASEDIR/smarts/local/conf/imk.dat.

SM_CLIENTCONNECT

This environment variable enables you to specify an alternate location for the clientConnect.conf client-side authorization file. You must specify the full path, including the filename. The default location is the BASEDIR/smarts/local/conf/clientConnect.conf directory.

SM_OKLOGIN_LOGLEVEL

This environment variable enables the logging of successful logins at ERROR severity, rather than NOTICE.

SM_OKLOGIN_LOGLEVEL is a Boolean.

SM_SERVERCONNECT

This environment variable enables you to specify an alternate location for the serverConnect.conf server-side authorization file. You must specify the full path, including the filename. The default location is the BASEDIR/smarts/local/conf/serverConnect.conf directory.

Variables that control date and time formatting

Managers and adapters format dates and times for printing in log files, archive files, and various messages. You can control many aspects of this formatting with the following variables.

Most dates and times displayed in SDK consoles are formatted by the consoles themselves and are not controlled by these variables.

Additional information can be found on many systems, such as:

- man pages on UNIX systems

SM_DATETIME_FORMAT

The SM_DATETIME_FORMAT environment variable determines the default value for the si_i18n_datetime_format() function. The si_i18n_datetime_format() function is the basis of CI_Time and CI_Exception. In addition, CI_Logger uses CI_Time classes to render times. Therefore, the SM_DATETIME_FORMAT environment variable determines the default rendering for any and all timestamps.

The SM_DATETIME_FORMAT environment variable determines the default rendering for any and all timestamps. The SM_DATETIME_FORMAT environment variable can be set to several different values. If you do not set the value of the SM_DATETIME_FORMAT variable, then the default is SI_DAT_LONG.

The following examples show the output, based on the setting.

```
SM_DATETIME_FORMAT=SI_DAT_LOCALE_NEUTRAL
  1969/12/31 20:00:42
SM_DATETIME_FORMAT=SI_DAT_SHORT
  31/12/69 20:00
SM_DATETIME_FORMAT=SI_DAT_MEDIUM
  31 déc. 1969 20:00:42
SM_DATETIME_FORMAT=SI_DAT_LONG
  31 décembre 1969 20:00:42 HMG-05:00
SM_DATETIME_FORMAT=SI_DAT_FULL
  mercredi 31 décembre 1969 20:00:42 États-Unis (New York)
SM_DATETIME_FORMAT=SI_DAT_DEFAULT
  31 décembre 1969 20:00:42 HMG-05:00
SM_DATETIME_FORMAT=
  31 décembre 1969 20:00:42 HMG-05:00
SM_DATETIME_FORMAT=bogus
```

31 décembre 1969 20:00:42 HMG-05:00

TZ

This system environment variable determines which time zone is used when formatting a time. As such, it controls the conversion from an internal time base to the appropriate local time. The syntax is TZ=NNNshh:mmDDD, where:

- NNN is the time zone name.
- s is an optional sign.
- hh and mm are the offset in hours and minutes from GMT. Positive offsets represent time zones behind GMT, negative offsets represent those ahead of GMT.
- DDD is the corresponding name of Daylight Savings time when it is in effect.

  For example, in the eastern United States, TZ is set to EST5EDT - eastern standard time/eastern daylight time, where EST is five hours behind GMT.

  All UNIX systems use a system initialization file to set this variable.

LC_TIME

This system environment variable controls the formatting of dates and times. By default, Managers use time and date formats that are sensitive to locale settings. For example, if you chose a date format that includes the name of the day of the week, in the English locale, the first weekday is formatted as Monday, while in the French locale it is formatted as Lundi.

If LC_TIME is not set, the system variable LC_ALL is used. If LC_ALL is not set either, the system variable LANG is used. Finally, if LANG is not set, a default “C locale” (essentially US English) is used.

The locale names and available locales are system specific.

Variables that control how VMware programs read and write files

These variables control the location of SDK output, such as log files and repository. They also control the location from which SDK programs read files, such as ASL scripts.

SM_BACKUP_FILE_LIMIT

SDK programs create a new log file when they are restarted or when the log file is rolled over. This environment variable determines the number of log files that are saved by SDK programs. Chapter 7 Managing Log Files provides additional information, including an example.

If this environment variable is set to 0, only the current log file and a backup of the previous log file are saved. If this variable is set to an invalid value, a warning message is written to the log file and the default value is used instead. Valid values are 0 through 999; the default value is 3.

SM_WRITEABLE

This environment variable specifies the location where SDK software writes output files such as repositories, log files, and saved consoles. By default, the location is set to BASEDIR/smarts/local. If you set the SM_WRITEABLE variable, you must also create the underlying directories.

Directories controlled by SM_WRITEABLE lists the directories whose location is controlled by SM_WRITEABLE:

Table 8-5. Directories controlled by SM_WRITEABLE

Subdirectory	Contents
/consoles	Saved consoles
/logs	Log files
/repos	Repository files

The location where log files are saved can be separately controlled with the SM_LOGFILES environment variable.

SM_LOGFILES

This environment variable is used to specify the location where log files should be written. Setting this variable is optional. If there is no value assigned to this variable, the log files will be written to the location SM_WRITEABLE/logs. If you set SM_LOGFILES, you must also create the /logs directory at the location specified by the variable.

This variable takes precedence over the value of the SM_WRITEABLE environment variable.

SM_RULESET_PATH

By default, SDK programs only invoke ASL scripts located in the BASEDIR/smarts/rules directory, the BASEDIR/smarts/local/rules directory, or a subdirectory of one of these directories. You can use this environment variable to specify additional locations from which SDK programs can invoke ASL scripts.

When set, the directories specified by SM_RULESET_PATH are searched before the system defaults.

Variables that control thread locks

These variables control thread locking in SDK programs.

SM_COUNT_REPOS_LOCKS

A Manager crashes whenever the stack used by a thread exceeds the space allocated. Threads acquire locks to access instances in the Manager's repository. If a thread exceeds a certain number of locks, then the lock fails with an error code of ELOOP and a message is written to the log file.

This environment variable is used to control lock counting. SM_COUNT_REPOS_LOCKS must be set before starting the Manager. By default, the lock counting is turned off. To enable lock counting, set SM_COUNT_REPOS_LOCKS to TRUE, yes, or 1.

SM_REPOS_LOCKS_LIMIT

This environment variable determines the number of locks which, if exceeded, causes the ELOOP error. SM_REPOS_LOCKS_LIMIT must be an integer number. The default value is 50.

VMware recommends a value that is less than 100. Setting low values might cause legitimate requests to fail and affect analysis. Setting high values might allow the thread to run out of stack space before reaching the lock count limit.

Wildcards Used by VMware Software

9

Types of wildcard patterns

A wildcard pattern is a series of characters that are matched against incoming character strings. You can use these patterns when you define pattern matching criteria.

Matching is done strictly from left to right, one character or basic wildcard pattern at a time. Basic wildcard patterns are defined in Basic wildcard patterns. Characters that are not part of match constructs match themselves. The pattern and the incoming string must match completely. For example, the pattern abcd does not match the input abcde or abc.

A compound wildcard pattern consists of one or more basic wildcard patterns separated by ampersand (&) or tilde (~) characters. A compound wildcard pattern is matched by attempting to match each of its component basic wildcard patterns against the entire input string. Compound wildcard patterns are listed in Compound wildcard patterns.

If the first character of a compound wildcard pattern is an ampersand (&) or tilde (~) character, the compound is interpreted as if an asterisk (*) appeared at the beginning of the pattern. For example, the pattern ~*[0-9]* matches any string not containing any digits. A trailing instance of an ampersand character (&) can only match the empty string. A trailing instance of a tilde character (~) can be read as “except for the empty string.”

Note Spaces are interpreted as characters and are subject to matching even if they are adjacent to operators like “&”.

Table 9-1. Basic wildcard patterns

Character	Description
?	Matches any single character. For example, server?.example.com matches server3.example.com and serverB.example.com, but not server10.example.com.
*	Matches an arbitrary string of characters. The string can be empty. For example, server*.example.com matches server-ny.example.com and server.example.com (an empty match).

Table 9-1. Basic wildcard patterns (Continued)

Character	Description
[set]	Matches any single character that appears within [set]; or, if the first character of [set] is (^), any single character that is *not* in the set. A hyphen (-) within [set] indicates a range, so that [a-d] is equivalent to [abcd]. The character before the hyphen (-) must precede the character after it or the range will be empty. The character (^) in any position except the first, or a hyphen (-) at the first or last position, has no special meaning. For example, server[789-].example.com matches server7.example.com through server9.example.com, but not server6.example.com. It also matches server-.example.com. For example, server[^12].example.com does not match server1.example.com or server2.example.com, but will match server8.example.com.
<n1-n2>	Matches numbers in a given range. Both *n1* and *n2* must be strings of digits, which represent nonnegative integer values. The matching characters are a non-empty string of digits whose value, as a nonnegative integer, is greater than or equal to *n1* and less than or equal to *n2*. If either end of the range is omitted, no limitation is placed on the accepted number. For example, 98.49.<1-100>.10 matches a range of IPv4 addresses from 98.49.1.10 through 98.49.100.10. Example of an omitted high end of the range: <50-> matches any string of digits with a value greater than or equal to 50. Example of an omitted low end of the range: <-150> matches any value between zero and 150. For a more subtle example: The pattern <1-10>* matches 1, 2, up through 10, with * matching no characters. Similarly, it matches strings like 9x, with * matching the trailing x. However, it does not match 11, because <1-10> always extracts the longest possible string of digits (11) and then matches only if the number it represents is in range.
|	Matches alternatives. For example,”ab|bc|cd” without spaces matches exactly the three following strings: “ab”, “bc”, and “cd”. A | as the first or last character of a pattern accepts an empty string as a match. Example with spaces “ab | bc” matches the strings “ab” and “ bc”.
\	Removes the special status, if any, of the following character. Backslash (\) has no special meaning within a set ([set]) or range (<n1-n2>) construct.

Special characters for compound wildcard patterns are summarized in Compound wildcard patterns.

Character	Description
&	“And Also” for a compound wildcard pattern. If a component basic wildcard pattern is preceded by & (or is the first basic wildcard pattern in the compound wildcard pattern), it *must* successfully match. Example: *NY*&*Router* matches all strings that contain NY and also contain Router. Example: <1-100>&*[02468] matches even numbers between 1 and 100 inclusive. The <1-100> component only passes numbers in the correct range and the *[02468] component only passes numbers that end in an even digit. Example: *A*|*B*&*C* matches strings that contain either an A or a B, and also contain a C.
~	“Except” for a compound wildcard pattern (opposite function of &).If a component basic wildcard pattern is preceded by ~, it *must not* match. Example: 10.20.30.*~10.20.30.50 matches all devices on network 10.20.30 except 10.20.30.50. Example: *Router*~*Cisco*&*10.20.30.*~10.20.30.<10-20>* matches a Router, except a Cisco router, with an address on network 10.20.30, except not 10.20.30.10 through 10.20.30.20.