Oracle® Communications Convergent Charging Controller

Service Management System Technical Guide

Release 6.0

May 2016

Copyright

Contents

About This Document

Scope

The scope of this document includes all the information required to install, configure, and administer the Service Management System application.

Audience

This guide was written primarily for system administrators and persons installing, configuring, implementing and administering the USMS application. The documentation assumes that the person using this guide has a good technical knowledge of the system.

Prerequisites

Although there are no prerequisites for using this guide, familiarity with the target platform would be an advantage.

A solid understanding of Unix and a familiarity with IN concepts are an essential prerequisite for safely using the information contained in this technical guide. Attempting to install, remove, configure or otherwise alter the described system without the appropriate background skills, could cause damage to the system; including temporary or permanent incorrect operation, loss of service, and may render your system beyond recovery.

This manual describes system tasks that should only be carried out by suitably trained operators.

Related Documents

The following documents are related to this document:

- *Service Management System User's Guide*
- SC3.1 Data Service for OPS/RAC http://download.oracle.com/sunalerts/1000611.1.html
- Step-by-Step Installation of 9i RAC on Sun Cluster v3 on https://support.oracle.com (doc ID 175465.1)
- Installing and Configuring Sun Cluster Software
- Xerlin XML editor on http://www.xerlin.org

Document Conventions

Typographical Conventions

The following terms and typographical conventions are used in the Oracle Communications Convergent Charging Controller documentation.

Formatting Convention	**Type of Information**
Special Bold	Items you must select, such as names of tabs. Names of database tables and fields.
Italics	Name of a document, chapter, topic or other publication. Emphasis within text.
Button	The name of a button to click or a key to press. **Example:** To close the window, either click **Close**, or press **Esc**.
Key+Key	Key combinations for which the user must press and hold down one key and then press another. Example: **Ctrl+P** or **Alt+F4**.
`Monospace`	Examples of code or standard output.
`Monospace Bold`	Text that you must enter.
variable	Used to indicate variables or text that should be replaced with an actual value.
menu option > menu option >	Used to indicate the cascading menu option to be selected. Example: **Operator Functions > Report Functions**
hypertext link	Used to indicate a hypertext link.

Specialized terms and acronyms are defined in the glossary at the end of this guide.

Chapter 1

System Overview

Overview

Introduction

This chapter provides a high-level overview of the application. It explains the basic functionality of the system and lists the main components.

It is not intended to advise on any specific Oracle Communications Convergent Charging Controller network or service implications of the product.

In this Chapter

This chapter contains the following topics.

What is the Service Management System?

Description

The Service Management System (SMS) product provides service management support for existing Oracle Communications Convergent Charging Controller Intelligent Network (IN) products.

The primary function of SMS is to provide operators with access to data used by service logic applications.

SMS provides the following:

- A central repository for other IN services, such as ACS and CCS
- Generic functions

The SMS main menu provides access to all installed services. To access any service, select the item from this menu.

Functions

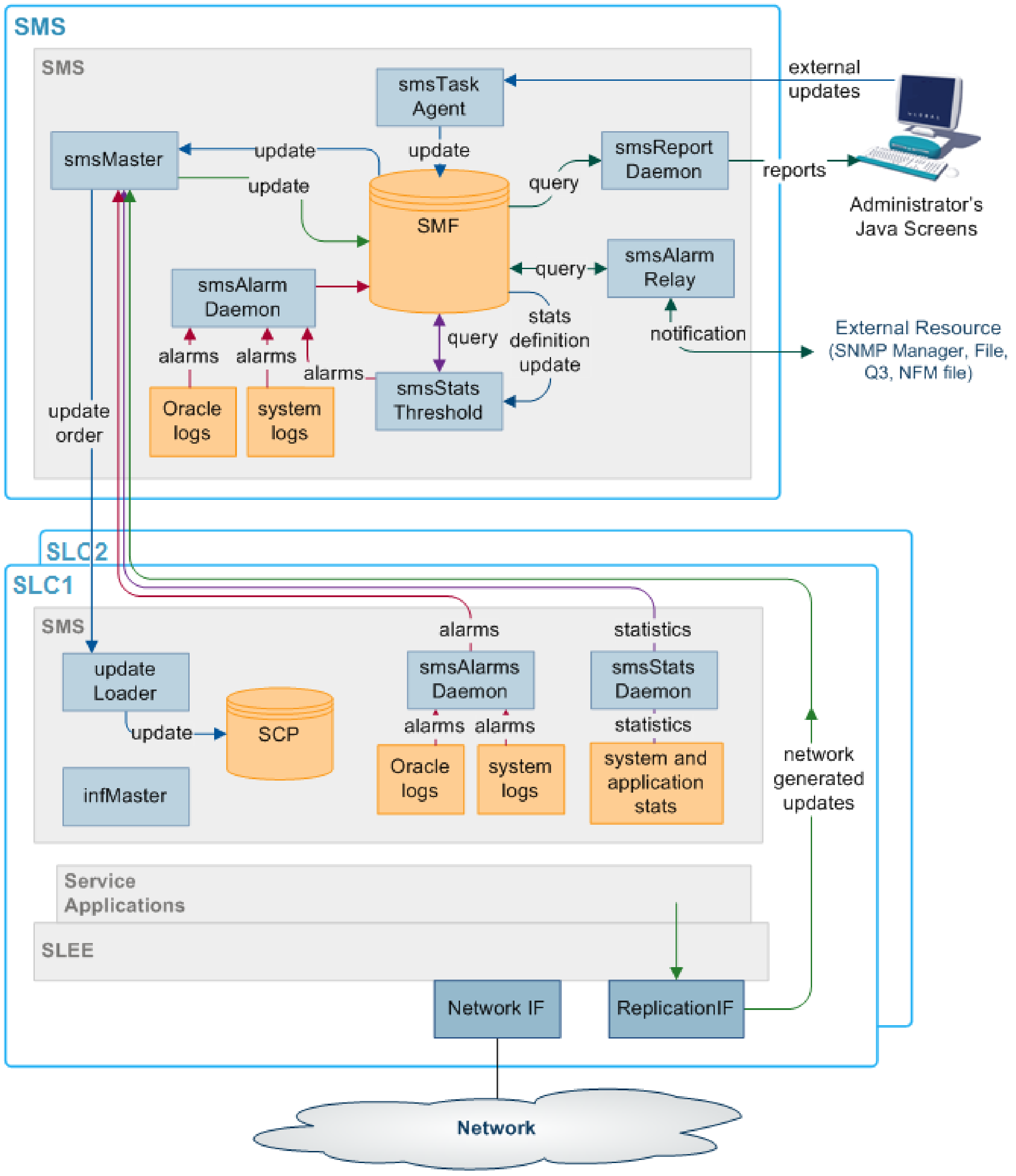
The generic functions of SMS include:

- Security
- Replication
- Statistics gathering
- Alarm Management
- Report generation

- Auditing of Database Changes

SMS component diagram

Here is an example of the main components of the SMS system.



SMS subsystems

There are four main subsystems within SMS:

- Replication
- configuration management
- Reporting functions

- File transfer

Replication

Replication provides the main method of transferring data around the Service Management System. It provides:

- A reliable and fault tolerant delivery of data:
  - From administrators and SLCs into the SMF
  - Changes to persistent data held in the SMF to all relevant SLCs (so all parts of the system have consistent data)
- Alternative network routing between the SLC and SMS under network failure, or buffered updates under complete network failure or SMS downtime
- Disaster recovery in the event of a network failure

Replication moves the following data:

- Configuration data for the smsStatsDaemon
- Configuration data for other installed IN software (such as ACS, CCS and VWS)
- Any update of application data due to the actions of the service running on the SLCs (including client account and call routing data)
- System, application and interface statistics
- Alarms

For more information about replication, see *Replication Overview* (on page 19).

Reporting Functions

The reporting functions enable the administrator to run reports against the data collected in the SMF.

The reports are configured in the SMS Java administration screens.

Data flow

There are two main methods of data transfer:

- Replication
- File transfer (using ftp)

Process Descriptions

This table describes the main components in SMS.

Process	Role	Further information
smsMaster	Receives update requests and forwards them to the SMF.	*smsMaster* (on page 120)
SMF	The main SMF on the SMS.	
SCP	The databases on the SLCs. They hold a subset of the data on the SMF.	
updateLoaders	Receives update orders from the smsMaster and inserts them into the SCPs.	*updateLoader* (on page 168)
Update Requesters	Update requesters run on SMSs and SLCs and may run on other IPs as well . They send update requests to the smsMaster. They include the smsAlarmDaemon and the smsStatsDaemon.	

Process	Role	Further information
smsTaskAgent	Forwards administrator's instructions to the smsMaster.	*smsTaskAgent* (on page 139)
smsAlarmDaemon	Collects alarms from local sources and forwards them to the smsMaster.	*Alarms* (on page 9)
smsAlarmRelay	Monitors the alarms in the SMF and forwards alarms to administrators.	*Alarms* (on page 9)
smsReportsDaemon	Enables the user to run reports against the data held in the SMF.	*smsReportsDaemon* (on page 122)
smsStatsDaemon	Collects statistics and forwards them to the smsMaster.	*Statistics* (on page 12)

Platform Configuration

Overview

There are three configurations that SMS can be installed on. They are:

- On a single platform
- With one SMS on one platform and one or more SLCs on separate platforms
- With multiple SMSs connected to a RAID array and one or more SLCs on separate platforms

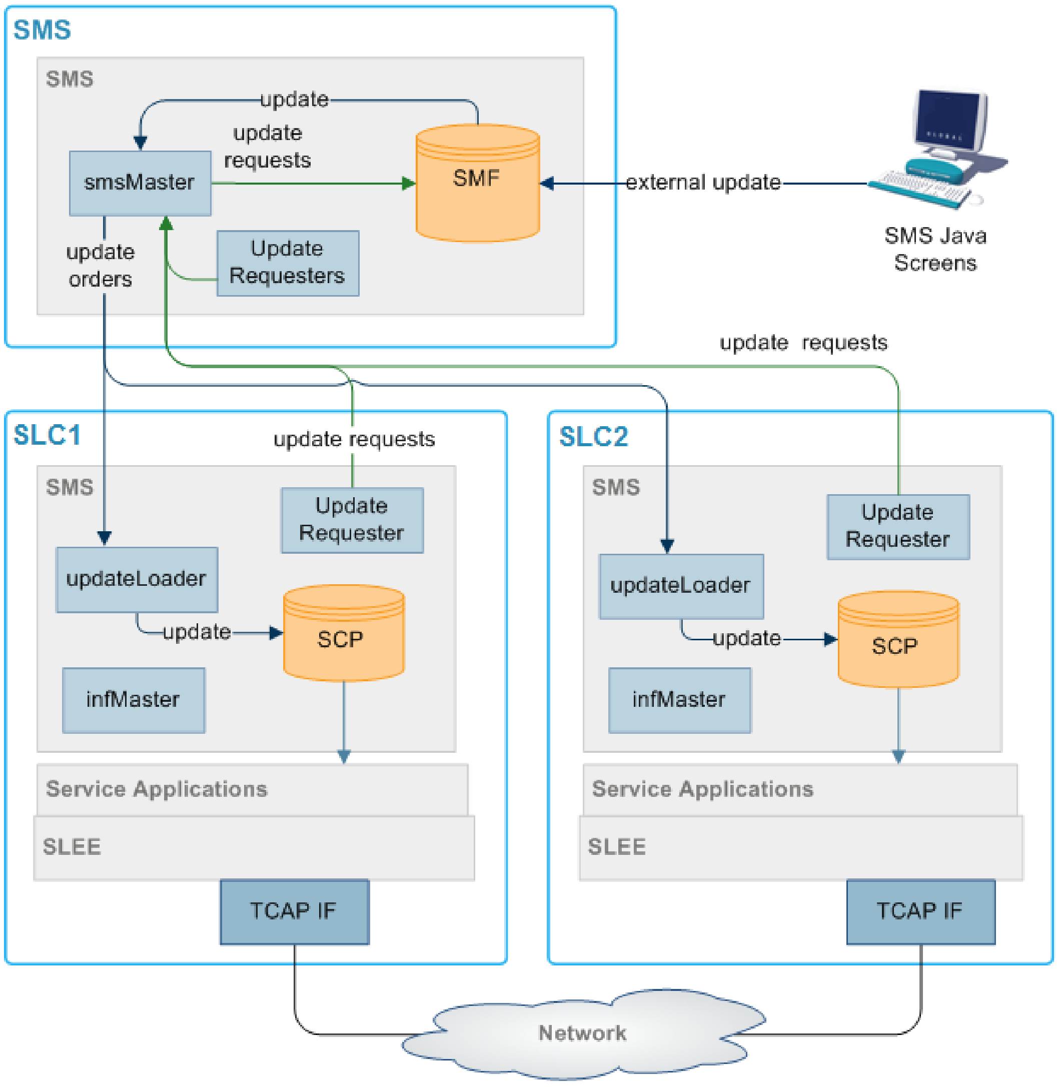
Unclustered platform configuration

Using the unclustered platform configuration, the smsSms package is installed on the SMS. The smsScp package is installed on one or more SLCs.

This configuration provides resilience by using a failover system from the SMS to the SLCs. However, while the SMS is unavailable, no configuration updates can be forwarded to the SLCs.

Unclustered platform configuration diagram

Here is an example of replication in an unclustered installation.



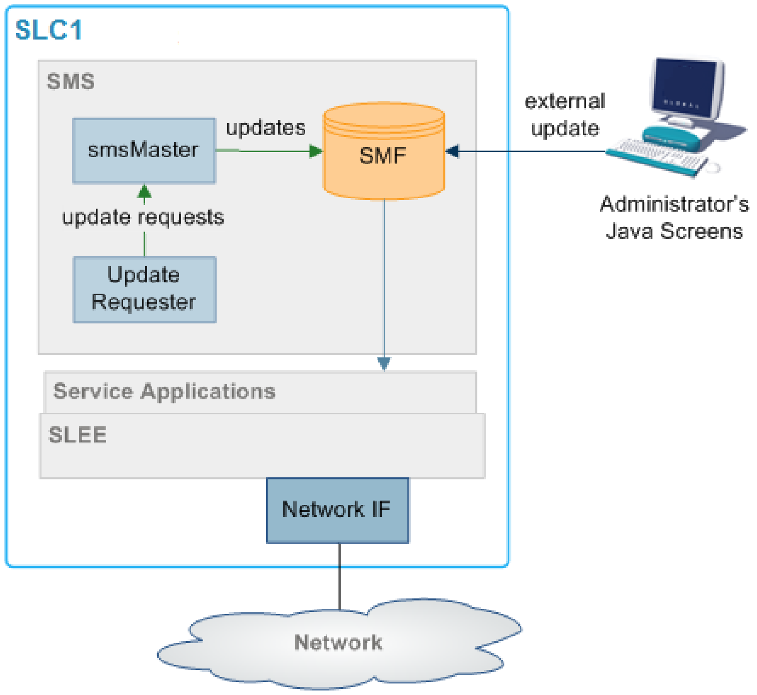
Single platform configuration

Using the single platform configuration, all required SMS functionality is installed on a single platform. Because all SMS functionality is on a single computer the parts of SMS which are involved in connecting the different components are removed.

This results in a simple, easy to administer system. However, because the system runs on one machine, resilience is reduced.

Single platform component diagram

Here is an example of the components in SMS installed on a single platform.



Maintaining Network Connections

Introduction

All replication elements (nodes) establish TCP connections with a master replicator by implicitly connecting to one.

To maintain reliable connection between nodes of the replication system two methods are employed to strengthen the underlying TCP protocol to be used:

- Heartbeating
- Dual network connection

Heartbeating

A simple heartbeating mechanism is used to overcome TCP's failure to detect connection severance (for example, cable failure).

Every node connected to a superior master node sends a periodic heartbeat message to which the master responds with an acknowledgment. This ensures both ends of the connection can detect failure within one heartbeat period.

When a connection fails, the **connecting** element should attempt to reconnect to a superior master. If the superior master is part of a cluster, the connecting element attempts to connect to the next master in the cluster.

Dual Network Connection

The replication system supports dual network connection to overcome a potential single point of failure (that of the underlying transport medium).

Each node can have two addresses by which it can be reached: a primary and a secondary address. These addresses can (and should) be on separate networks.

When a connection to a superior master is required by an element, two connection attempts are made:

1 Primary address (over the primary network)
2 Secondary address (over a secondary network)

The replication element uses the first connection to succeed, closing the other connection first.

If required, a configurable delay (of up to one second) occurs between the connection attempt to the primary address and the secondary one.

This provides the ability to favor the primary network over the secondary (for example, if one network has a better known latency).

If no delay is configured, the connection attempts occur simultaneously. If both networks have similar latency, the one that ultimately gets used is unpredictable.

smsTrigDaemon

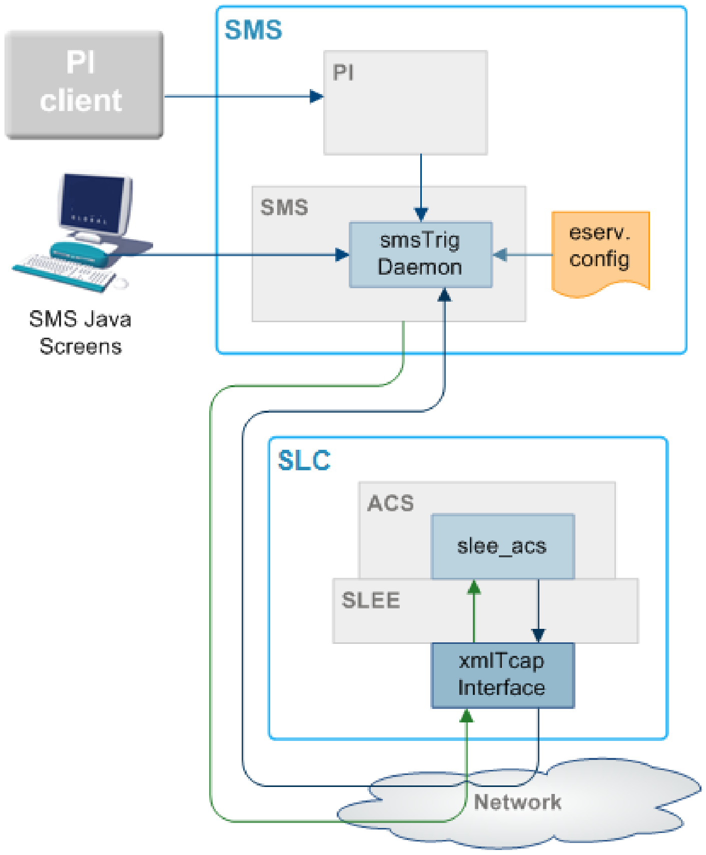
Purpose

smsTrigDaemon manages control plan execution requests. It runs on the SMS platform.

smsTrigDaemon accepts control plan execution requests from either a remote PI client or the Java management screens. It forwards requests to ACS through the xmlTcapInterface on the SLC platform. An indication of whether or not the requests were successful passes back from the ACS to the initiating client.

Architectural overview

This diagram shows smsTrigDaemon and components that surround it.



Message flows

This table describes message the message flows that smsTrigDaemon uses.

Stage	Description
1	The Java management screens send control plan execution requests to smsTrigDaemon over a CORBA transport layer. Each request contains the name of the control plan to be executed, the SLC service handle, the CLI of the subscriber against which the control plan should be executed, an optional called party number and extensions.
2	A remote PI client sends control plan execution to requests to the provisioning interface. As with stage 1, each request contains the name of the control plan to be executed, the SLC service handle, the CLI of the subscriber against which the control plan should be executed, an optional called party number and extensions.
3	The provisioning interface forwards requests to smsTrigDaemon over the FIFO layer transport layer.
4	Using an XML request, smsTrigDaemon forwards the control plan execution request to the xmlTcapInterface on the SLC platform.
5	The xmlTcapInterface constructs an InitialDP and sends it to ACS through the SLEE. For more information about the SLEE, see *Service Logic Execution Environment Technical Guide*.
6	An indication of success or failure is returned to the xmlTcapInterface using a Connect, Continue or ReleaseCall component.

Stage	Description
7	The indication of success or failure is sent to smsTrigDaemon using an HTTP response. smsTrigDaemon then sends the indication back to the client.

Note: Third parties can also send XML requests directly to the xmlTcapInterface.

Components

The smsTrigDaemon interacts with three subsystems:

- Provisioning Interface
- xmlTcapInterface
- SLEE

PI

The Provisioning Interface (PI) provides a mechanism for manipulating data in the SMF. It enables bulk or scripted operations on SMF data where manual input using the Java management screens would be inefficient.

For more information, see *PI User's and Technical Guide*.

xmlTcapInterface

The xmlTcapInterface enables the SLEE to inter-work with a TCAP protocol. The interface converts XML messages arriving from smsTrigDaemon into SLEE events. Similarly, the interface converts events arriving from the SLEE into XML messages that smsTrigDaemon understands.

For more information, see *XML TCAP Interface Technical Guide*.

Alarms

Introduction

Alarms from the SMS and SLCs are collected in the SMF using replication. A set of tools enable management of the alarms in the SMF. Functions include:

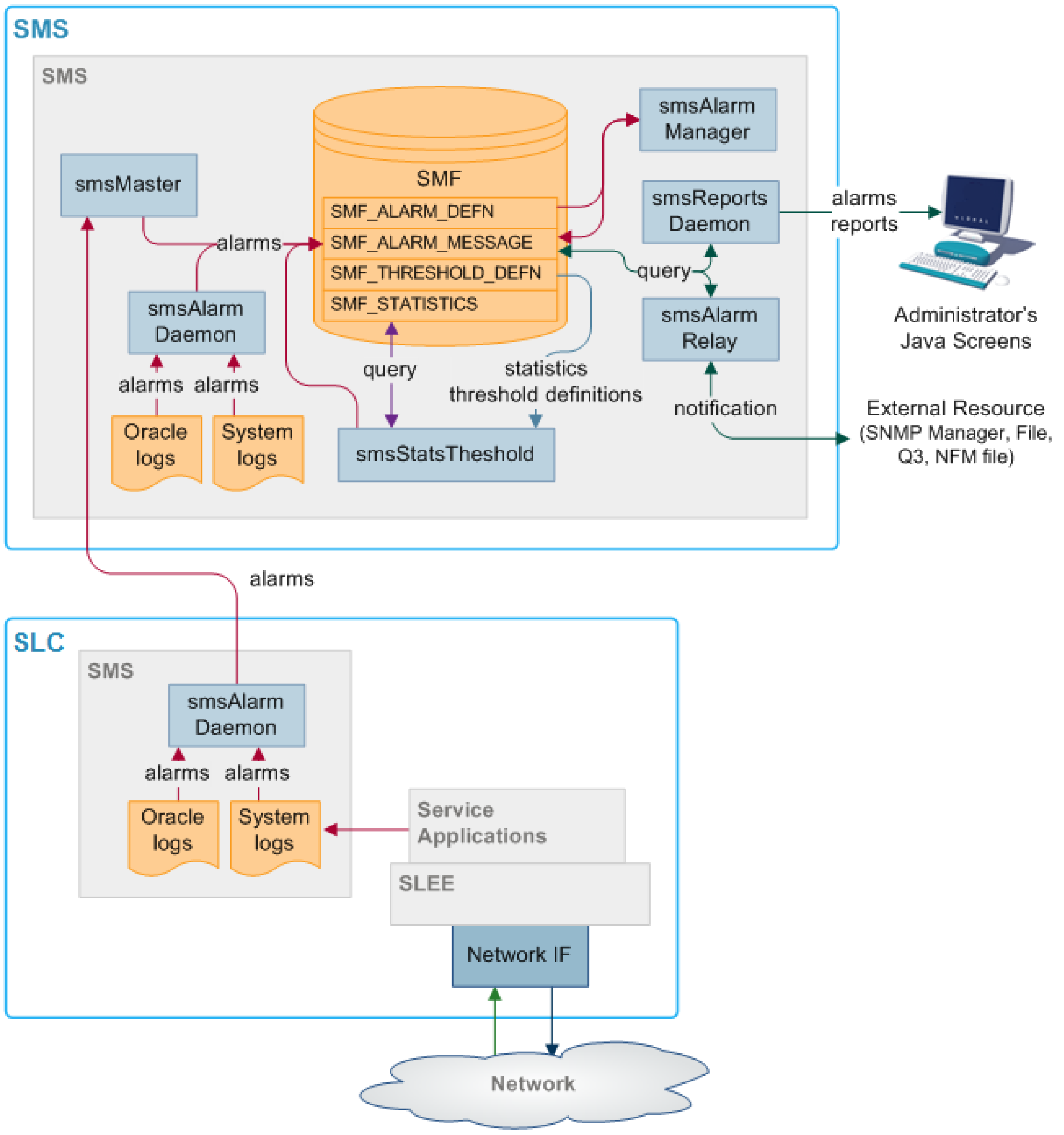
- Filtering alarms
- Setting notification destinations
- Monitoring

This functionality is configured using the alarms screens in SMS. For more information about configuring alarms, see *Service Management System User's Guide*.

Alarms can be generated from monitoring statistics.

Alarms diagram

Here is an example of the alarms transfer process.



Alarms replication process

This table describes the stages involved in collecting and reporting about alarms within the SMS system using replication.

Stage	Description
1	Alarms are collected by the smsAlarmDaemon on the SMS and SLCs. Sources include: • The syslog file • Oracle logs
2	The smsAlarmDaemon sends an update request to the superior master (usually the smsMaster). **Exception:** The smsAlarmDaemon on the SMS makes its updates directly to the SMF, without sending anything to the smsMaster.

Stage	Description
3	When the superior master receives an update request, it inserts the updated data into the SMF_ALARMS_MESSAGE table of the SMF.
4	The smsAlarmManager matches each alarm instance in the SMF_ALARMS_MESSAGE table with the correct alarm type from the SMF_ALARM_DEFN table, and additional information about the alarm type is saved with the alarm instance.
5	The smsAlarmRelay process monitors the SMF_ALARMS_MESSAGE table and forwards alarms to the specified external resource. **Note:** The administrator can run reports on the collected alarms using the reports screens in SMS (which are executed by the smsReportsDaemon).

Statistics thresholds

Alarms can be generated from specific statistical measures.

The smsStatsThreshold process monitors the SMF_STATISTICS table in the SMF database. When a statistic or statistics match a rule specified in the SMF_THRESHOLD_DEFN table, the smsStatsThreshold process inserts an alarm record into the SMF_ALARM_MESSAGE table in the SMF database.

For more information about configuring statistics thresholds, see *Service Management System User's Guide*.

Enhanced Fault Management

Enhanced Fault Management (EFM) takes the alarms that are produced by the system and matches alarm instances to information that is held in the database for each alarm type. The alarm instances, including the additional information can then be relayed to an external resource for further processing.

Description of processes and executables

This table describes the roles of the components involved in the alarms process.

Process	Role	Further information
smsAlarmDaemon	Collects alarms from local sources and forwards them to the smsMaster.	*smsAlarmDaemon* (on page 95)
smsMaster	Receives alarms from smsAlarmDaemons and forwards them to the SMF.	*smsMaster* (on page 120)
smsAlarmRelay	Monitors the SMF_ALARM_MESSAGE table in the SMF and forwards alarms to relevant notification points (including SNMP).	*smsAlarmRelay* (on page 120)
smsReportsDaemon	Enables the user to run reports against the alarms held in the SMF.	*smsReportsDaemon* (on page 122)
smsStatsThreshold	Monitors the SMF_STATISTICS table in the SMF. If the statistics meet certain rules, the this process creates an alarm and inserts it into the SMF_ALARM_MESSAGE table in the SMF.	*smsStatsThreshold* (on page 137)
smsAlarmManager	The smsAlarmManager matches alarm instances with the alarm definitions stored in the SMF_ALARM_DEFN table on the SMF, and adds the extra information stored in the definition to each instance of that alarm as it occurs.	*smsAlarmManager* (on page 97)

Alarm replication and buffering

The smsAlarmDaemon filters alarms before they are sent. This enables:

- Protection against the SMS being flooded with alarms
- Filtering of repeating alarms

For more information about buffering alarms, see *smsAlarmDaemon* (on page 95).

Statistics

Introduction

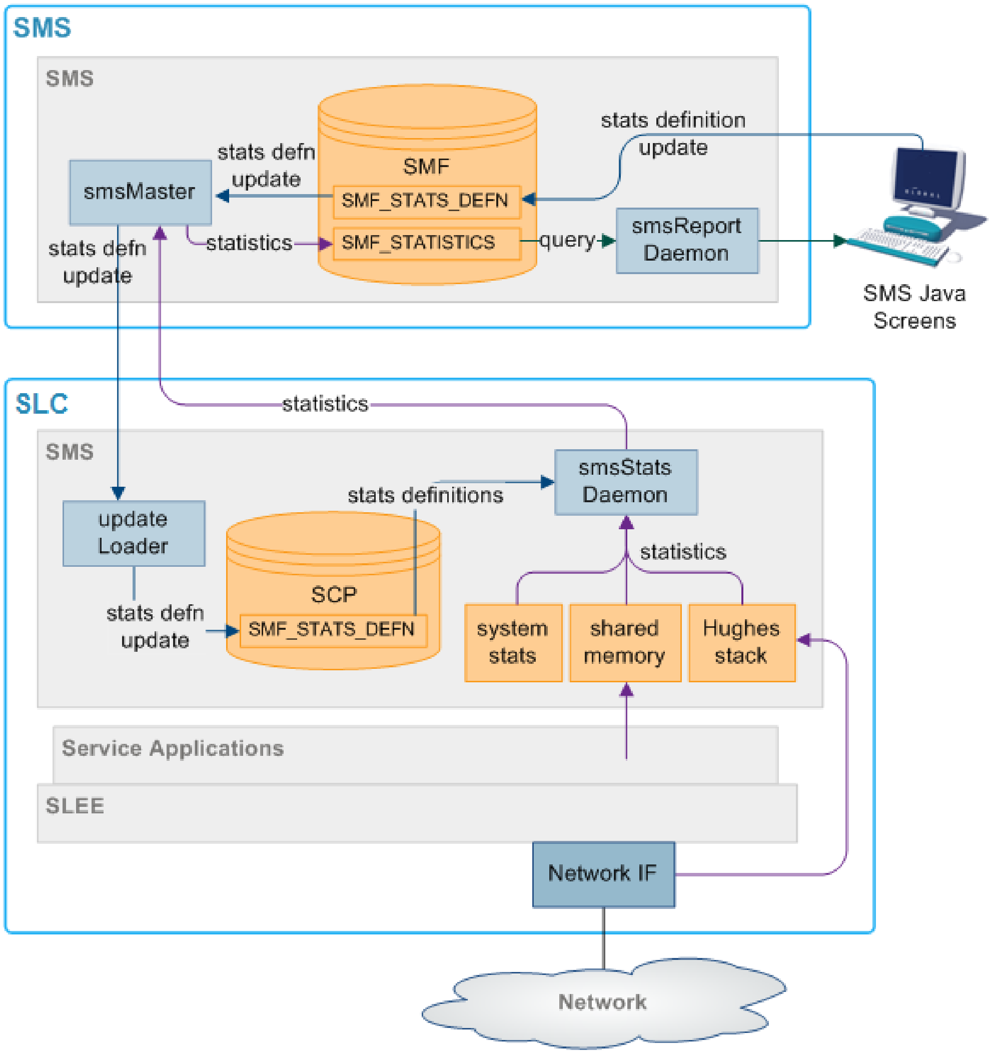
Statistics generated by the SMS and SLCs are collected in the SMF_STATISTICS table of the SMF database. A set of tools provides management functions. Functions include:

- Filtering statistics
- Setting rules for statistics thresholds which raise alarms
- Running reports against the statistics held in SMF_STATISTICS

For more information about using these functions, see *Service Management System User's Guide*.

Statistics collection diagram

Here is an example of the statistics collection process.



Description of processes and executables

This table describes the roles of the components involved in the statistics process.

Process	Role	Further information
smsStatsDaemon	Collects statistics from SLCs and forwards them to smsMaster.	*smsStatsDaemon* (on page 160).
smsMaster	Receives statistics from the smsStatsDaemons and forwards them to the smsMaster for insertion into the SMF.	*smsMaster* (on page 120).
smsReportsDaemon	Enables the user to run reports against the statistics held in the SMF.	*smsReportsDaemon* (on page 122).
smsStatsThreshold	Monitors the SMF_STATISTICS table in the SMF. If the statistics meet certain rules, the smsStatsThreshold process creates an alarm and forwards it to the smsAlarmDaemon on the SMS.	*smsStatsThreshold* (on page 137).

Statistics collection process

This table describes the stages involved in collecting statistics within the SMS system using replication.

Stage	Description
1	Statistics are gathered by the statistics daemon process (smsStatsDaemon) which runs on each SLC platform. Statistics which are collected include: • Statistics from the shared memory which are generated by the slee_acs • TCAP statistics from files saved by the TCAP interface • System statistics from the kernel
2	At regular intervals, the smsStatsDaemon sends the values to the smsMaster process on the SMS platform as an update request.
3	The smsMaster adds the new statistics to the SMF_STATISTICS table in the SMF.
4	The administrator can run reports on the collected statistics using the statistics screens in SMS (which are executed by the smsReportsDaemon).

Statistics thresholds

Alarms can be generated from specific statistical measures.

The smsStatsThreshold process monitors the SMF_STATISTICS table in the SMF database. When a statistic or statistics match a rule specified in the SMF_THRESHOLD_DEFN table, the smsStatsThreshold process inserts an alarm record into the SMF_ALARM_MESSAGE table in the SMF database.

For more information about configuring statistics thresholds, see *Service Management System User's Guide*.

Statistics collected

The statistics system can collect any SMS-compatible IN application statistics. These are typically coarse values related to the general performance and behavior of the application. Typical statistics values include:

- Total number of requests from SSF
- Number of call instances resulting in error treatment
- Number of calls from invalid geographical locations
- Number of calls reaching successful call completion to international locations
- Number of calls reaching successful call completion to international category one partners

Statistics sources may include:

- System statistics from the syslog
- System statistics from the operating system
- Statistics from the Sigtran stack
- Statistics from shared memory

Note: For statistics about call processing, see also *Advanced Control Services Technical Guide*.

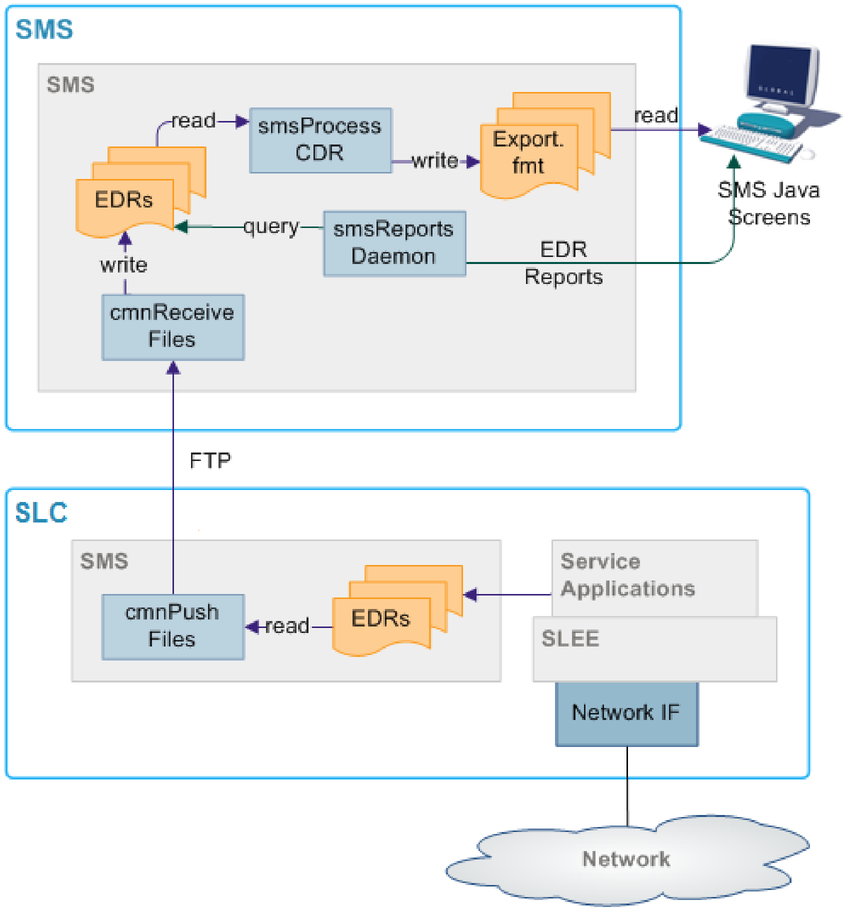
EDRs

Introduction

The SMS software provides a complete, integrated reporting mechanism for Event Detail Records (EDR). It allows the developers of SMS-compatible IN applications to add report functions to their product, through the SMS reports interface.

EDR file transfer diagram

Here is an example of the transfer of files between SLCs and the SMS.



EDR file transfer process

This table describes the stages involved in transferring files around the system using the Common File Transfer process. The files usually transferred are EDRs.

Stage	Description
1	On the SLCs, cmnPushFiles collects files from the configured input directory and transfers them to the configured output directory on the SMS through an stdout. It adds the destination directory to the file.
2	If the transfer fails, cmnPushFiles copies the files to the configured retry directory to attempt the transfer again later.

Stage	Description
3	When the files are successfully transferred to the SMS, cmnPushFiles moves the files to the configured completed directory.
4	On the SMS, cmnReceiveFiles scans the configured input directory and moves any files to the directory specified in the file.
5	smsCdrProcess.sh scans its input directory for ***.cdr** files and moves them to its processed directory.

Description of processes and executables

This table describes the roles of the components involved in the alarms process.

Process	Role	Further information
cmnPushFiles	Reads files from a specified directory and transfers them to the SMS using stdout. Depending on the success of the transfer, the file is also moved to another directory on the origination SLC.	*cmnPushFiles* (on page 150).
cmnReceiveFiles	Collects files from the input directory on the SMS and writes them to the specified output directory on the SMS.	*cmnReceiveFiles* (on page 94).
smsCdrProcess.sh	Provides a set of EDR processing and archiving functions.	*smsCdrProcess.sh* (on page 115).
smsReportDaemon	Enables the user to run reports against the statistics held in the SMF.	*smsReportsDaemon* (on page 122).

Directory structure and filenames

So that the Unix transfer scripts can locate the output EDR file, the file should be named according to the naming convention. This is usually done by the processes which create the files.

The directory structure which holds the files is in **/IN/service_packages/SMS/cdr/**.

For more information about the directory structure, see *Advanced Control Services Technical Guide*.

The file name is *ApplicationID***.cdr**. In this case, the complete specification of the currently active EDR filename for the ACS application is *APP_yyyymmddhhmmss***.txt**.

Where:

- *APP* is the three letter acronym for the originating process
- *yyyymmddhhmmss* is the date and time the file started to be written to

There is no need for the application to provide any further detail in the file name, as the subsequent processing of the EDR files can perform this. The file names for archived files on the SLC and SMS are detailed in the section that deals with the subsequent processing of these files.

EDR intermediate file format

The intermediate EDR, as output from the SMS EDR API is written to the **/IN/service_packages/SMS/cdr/current/** directory.

The format of the file is a **|** separated list of TAG=VALUE pairs, except for the first entry which is the service name followed by a **|**. Each record is new line separated.

Example:

```
# File created at 1999060312449
```

```
Acs_Service|SN=1800906420|TN=4770360|CGN=9380360|TCS=1999060312449
Acs_Service|SN=1800906421|TN=4770361|CGN=9380361|TCS=1999060312450
Acs_Service|SN=1800906422|TN=4770362|CGN=9380362|TCS=1999060312457
Acs_Service|SN=1800906423|TN=4770363|CGN=9380363|TCS=1999060312521
Acs_Service|SN=1800906424|TN=4770364|CGN=9380364|TCS=1999060312590
Acs_Service|SN=1800906425|CGN=9380365|TCS=1999060312449
Acs_Service|SN=1800906426|CGN=9380366|TCS=1999060312449
Acs_Service|SN=1800906427|TN=4770367|CGN=9380367|TCS=1999060313036
Acs_Service|SN=1800906428|TN=4770368|CGN=9380368|TCS=1999060312036
```

Chapter 2

Replication Overview

Overview

Introduction

This chapter explains the replication system used in SMS.

In this chapter

This chapter contains the following topics.

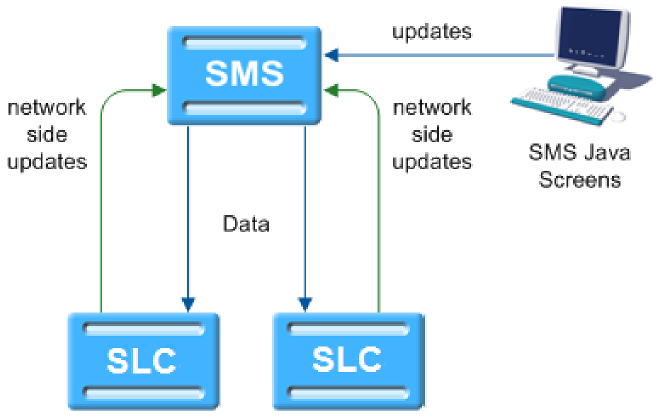
What is Replication?

Introduction

Replication is the system which transfers data between nodes in the IN installation.

Data flow

The SMF database on the SMS holds the full set of authoritative data within the system. Data required for call processing and resilience is forwarded to the SCP database on the SLCs using SMS replication. Updates are received from processes on the SMSs and the SLCs and from the Service Management System administration screens.



Replication process

This table describes the stages involved in replicating data around the system.

Stage	Description
1	Update Requests come from one of the following: • The administration screens • An event on the SMS or SLCs
2	If the update comes from the administration screens, one of the following occurs: • Forwarded to the smsTaskAgent, and then through to an smsMaster • Inserted directly into the SMF database If the update request comes from the SMS or SLCs, the relevant update requester sends an update request to an smsMaster (parent).
3	When an smsMaster (parent) receives an update request, it: a. Sends an update order to all configured destination replication groups (there may be no relevant groups, in which case no order is sent) b. Spawns a local smsMaster (child) process to insert the updated data into the SMF database.
4	updateLoader on the relevant SLCs reads the update order from the socket and inserts the data into the SCP database.
5	If requested to do so, updateLoader sends a confirmation to smsMaster that the update completed successfully.

Nodes

Replication occurs between nodes in the system. Nodes allow specific processes on machines to be replicated to and from, and for more than one node to exist on a single machine. Each node has a node number which identifies the node.

For more information about configuring nodes, see *Service Management System User's Guide*.

Superior Master Nodes

Superior master nodes are forwarded all data update requests within SMS, and distribute update orders to all SLCs that require the replication data through the updateLoaders.

In a clustered installation, the superior master role is shared between the available smsMaster nodes on the SMSs.

In an unclustered installation, the superior master node is the node with the lowest node number in the system. This is usually the smsMaster on the SMS, but at times may be an infMaster on an SLC.

Update Loader nodes

Update loader nodes run on any SLC that requires database updates. They are the updateLoader processes running on the SLCs. They accept update orders from superior master nodes and insert the data into the local SCP database.

The update loaders on a single SLC platform are independent of each other and are treated as separate replication nodes to the replication system. Hence there can be more than one per machine, although in practice there is normally just one.

An update loader must always be connected to a master. Even if it is not receiving any information from the master, it will have a connection.

Update Requester nodes

Update requesters create update requests in response to specific events on the SLCs and send them to the superior master to update the centralized data (and from there it is replicated to the relevant SLCs). Update requesters include:

- replicationIF
- smsAlarmsDaemons
- smsStatsDaemons

Update requesters do not need to be configured in the database.

Replication groups

A replication table has one or more replication groups. A replication group can be assigned to one or more replication nodes.

Example:

- Replication Group A resides on Node 1, Node 2 and Node 3
- Replication Group B resides on Node 1 and Node 3

Primary replication nodes

Primary nodes can be defined for a specific replication group. The primary is the highest priority destination node for the data defined in the replication group. This enables the IN to assign particular services to specific nodes, but still provide a failover to other nodes as required.

This only sets the node as the primary for the specific group involved and is independent of other groups. A node may be defined as a primary for one group without being a primary for another group.

Example:

- Replication Group A resides on Node 1, Node 2 and Node 3, where Node 3 is the primary for group A.
- Replication Group B resides on Node 1 and Node 3, where Node 1 is the primary for group B.

Primary nodes are not required unless a service is running with different priority on different nodes.

Update requests to primary nodes

Primary node status is relevant for processes which are requesting an smsMaster to update the SMF.

The update processes have three types of Update Requests:

1 Make the change and do not confirm that it has been made.
2 Send a notification when the change has been made to the SMF.
3 Send a notification when the change has been made to the primary replication node for this replication group.

The primary node status is used when the third type of update request is used. While the update may be successful without the primary node being configured, the requesting process may register errors if the notification of the update is not received.

For more information about setting primary and secondary status within a replication group, see *Service Management System User's Guide*.

Master Controllers

A master controller is any process which provides instructions to a superior master node. Possible instructions include:

- Update configuration
- Merge databases
- Resync databases

Master controllers include executables started from the command line and functions embedded in other processes. They include:

- smsTaskAgent
- resyncServer
- smsCompareResyncServer

Failover and Error Recovery

Introduction

If a node becomes unavailable for any reason, the system attempts to continue functioning. The nodes that remain available continue to operate normally. Updates for the node that is unavailable are queued for as long as the queue space lasts.

When the node becomes available again, the queued updates are resent.

If nodes become out of sync to the point where they cannot automatically recover, a manual resync can be run.

updateLoader failure

If the update loader fails, then the updates are queued until it is back on-line. If the Update Loader is still down after a period of time and a smsMaster's pending queue reaches its configured maximum size, then the update loader is marked as "Out Of History" by that smsMaster and its updates are removed. If this happens, after the Update Loader is back on-line, a total database re-synchronization is performed with the smsMaster.

Update queuing

If the nodes become disconnected, a number of processes queue updates until the connection is restored. After the connection is restored, the queued updates execute normally.

smsMaster queues all updates it sends out until an acknowledgment is sent out by the receiving updateLoader. The number of updates that are queued is set in the smsMaster configuration.

updateLoader queues all uncompleted updates in a file named using the following format:

*updateLoaderNodeNumber***-queuedOrders.dat**

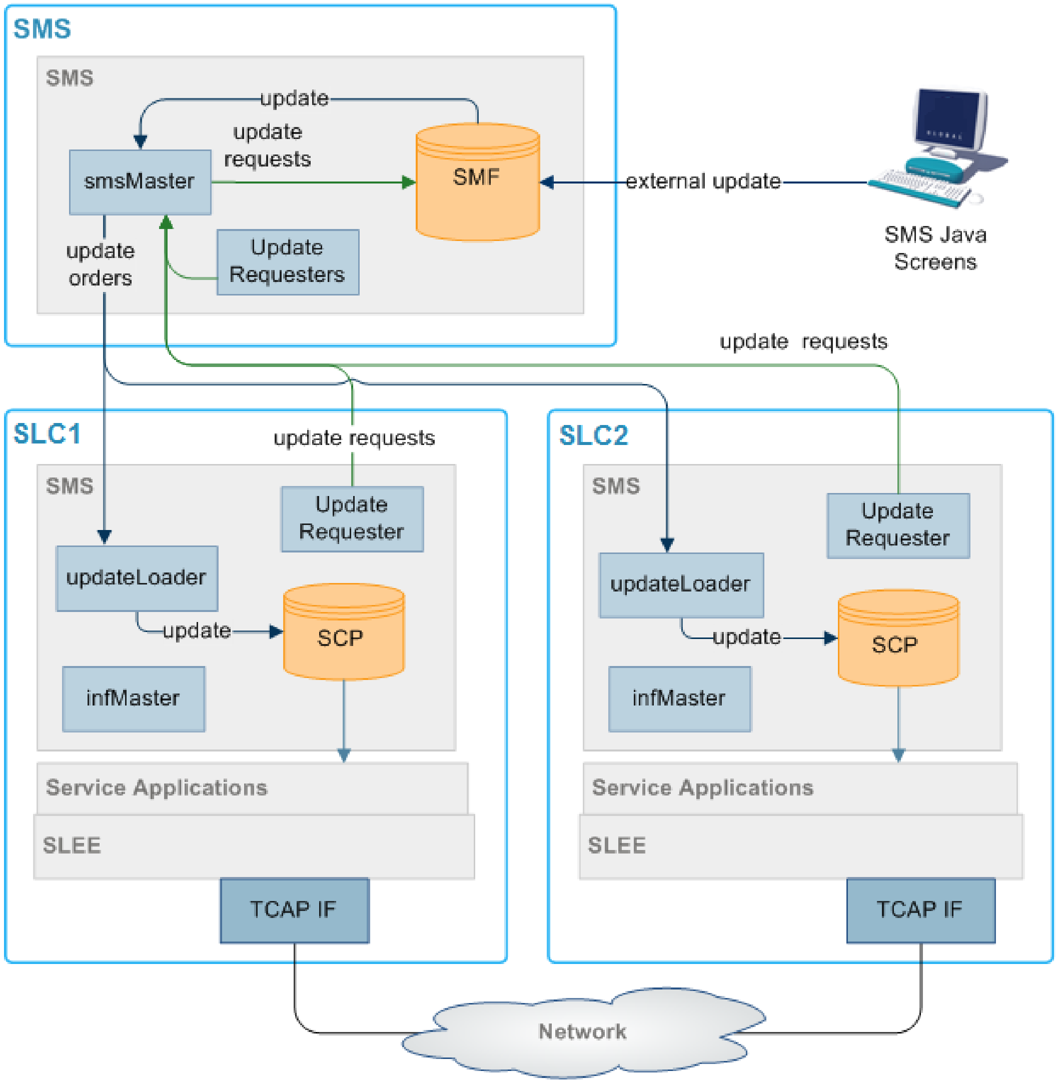
Further information

For more information on failover and error recovery processes, see *Replication Check* (on page 41).

Replication in an Unclustered Installation

Replication component diagram

Here is an example of replication in an unclustered installation.



Replication components

This table describes the components of replication in an unclustered installation.

Process	Role	Further information
smsMaster	Runs on the SMS handling updates throughout SMS. This is the superior master for all connected nodes.	*smsMaster* (on page 120)
infMaster	An infMaster runs on each SLC. If it becomes the node with the highest number of all connected nodes, it stands in as the superior master until a higher node number becomes available again.	*infMaster* (on page 155)

Process	Role	Further information
updateLoaders	An updateLoader runs on each SLC. It manages all incoming update orders and inserts updated data into the SCP. At any point in time, an updateLoader is connected to a specific superior master.	*updateLoader* (on page 168)
update requesters	update requesters may run on any machine. They send update requests to the Superior Master.	*Update Requester nodes* (on page 21)
smsMergeDaemon	The smsMergeDaemon runs on the SMS and monitors the connections between the SMS and the SLCs. If it notices a break in the connection, it may start a merge to update the disconnected nodes.	*smsMergeDaemon* (on page 119)
smsTaskAgent	The smsTaskAgent accepts instructions from the SMS Administration screens and produces instructions for the smsMaster. It generates the replication config file and copies it to the SLC nodes.	*smsTaskAgent* (on page 139)
smsNamingServer	The smsNamingServer enables non-SMS components to connect to elements within the SMS.	*smsNamingServer* (on page 121)
SMF	This Oracle database holds authoritative data for all SLCs.	
SCPs	These Oracle databases hold the subset of SMF data required to route calls.	

Updates

The replication system performs ‘row’ level updates and buffers updates to reduce processing load on the real-time system elements. This is achieved by holding the update requests in a memory resident queue (called the Pending Updates Queue) until replication has been successfully completed.

Update requests are performed in the order they arrive at the superior master.

Inferior Master Nodes

An inferior master node is a master node with a higher node number than that of the current superior master. It does not perform any function unless it becomes the available master node with the highest node number (in which case it becomes the superior master).

Node numbers

This table lists the node number ranges and their details for an unclustered installation.

Node Numbers	Description
1	This node number must assigned to the smsMaster process on the SMS.
17-255	These node numbers are available to infMaster processes on the SLCs.
256-511	These node numbers are available to updateLoaders on the SLCs.

Node Numbers	Description
512-999	These node numbers are available to updateRequesters. They are usually configured in the following pattern: • 601-699 Replication IF nodes • 701-799 smsStatsDaemon nodes • 801-899 smsAlarmDaemon nodes
1000	In an unclustered installation, this node number is used for the smsMergeDaemon.

Note: Node numbers are unique.

Failover

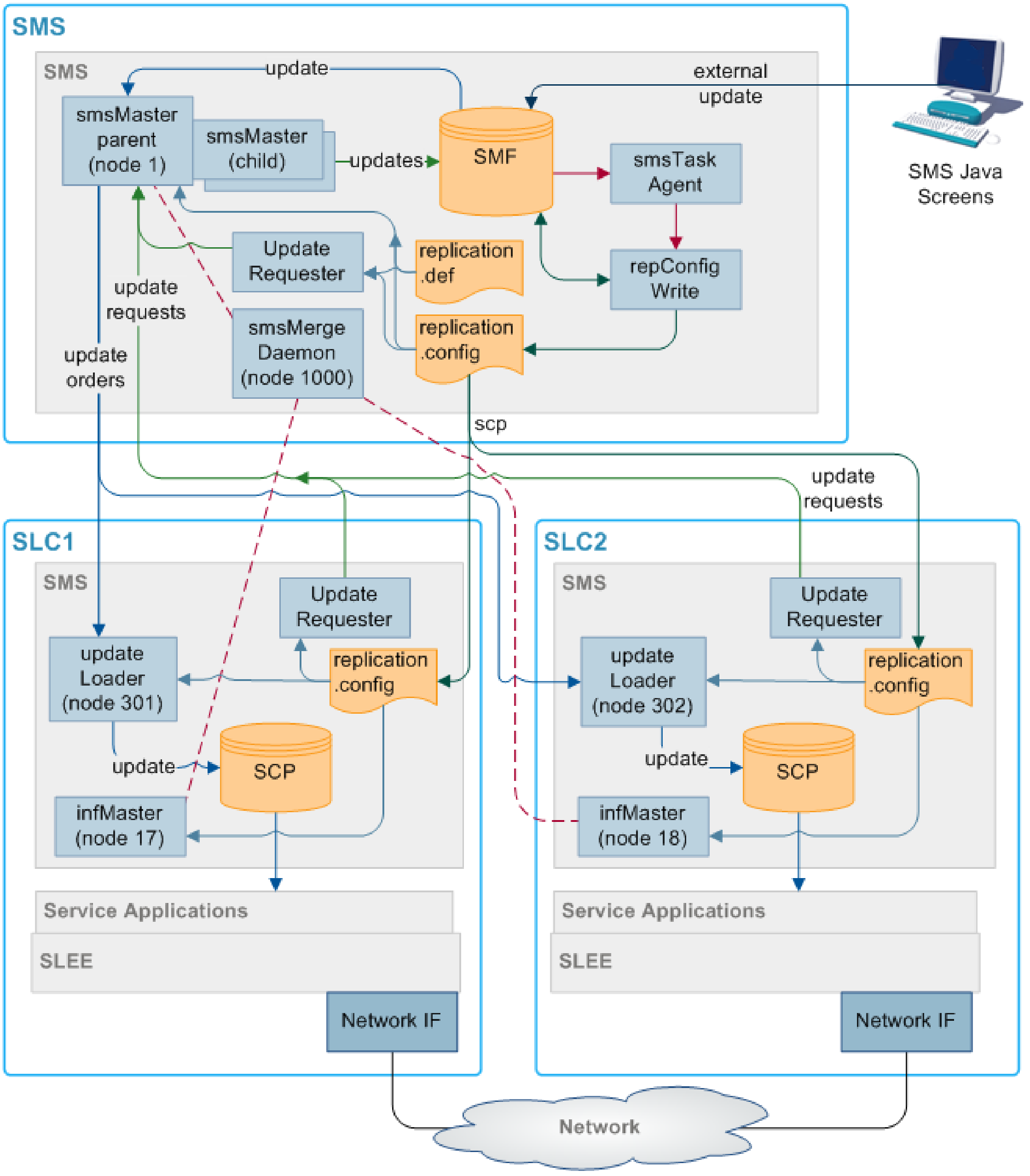
If a node becomes disconnected from the smsMaster node (due to network failure or a problem with the SMS), it attempts to contact the other nodes in descending node number order until it locates a node it can connect to.

An infMaster on one of the SLCs becomes the acting superior master until the failure is resolved. After the smsMaster becomes available again, smsMergeDaemon instructs the infMaster to merge its updates with the smsMaster.

If the infMaster that is the acting superior master becomes unavailable before the smsMaster is available again, the infMaster with the next node number is used instead.

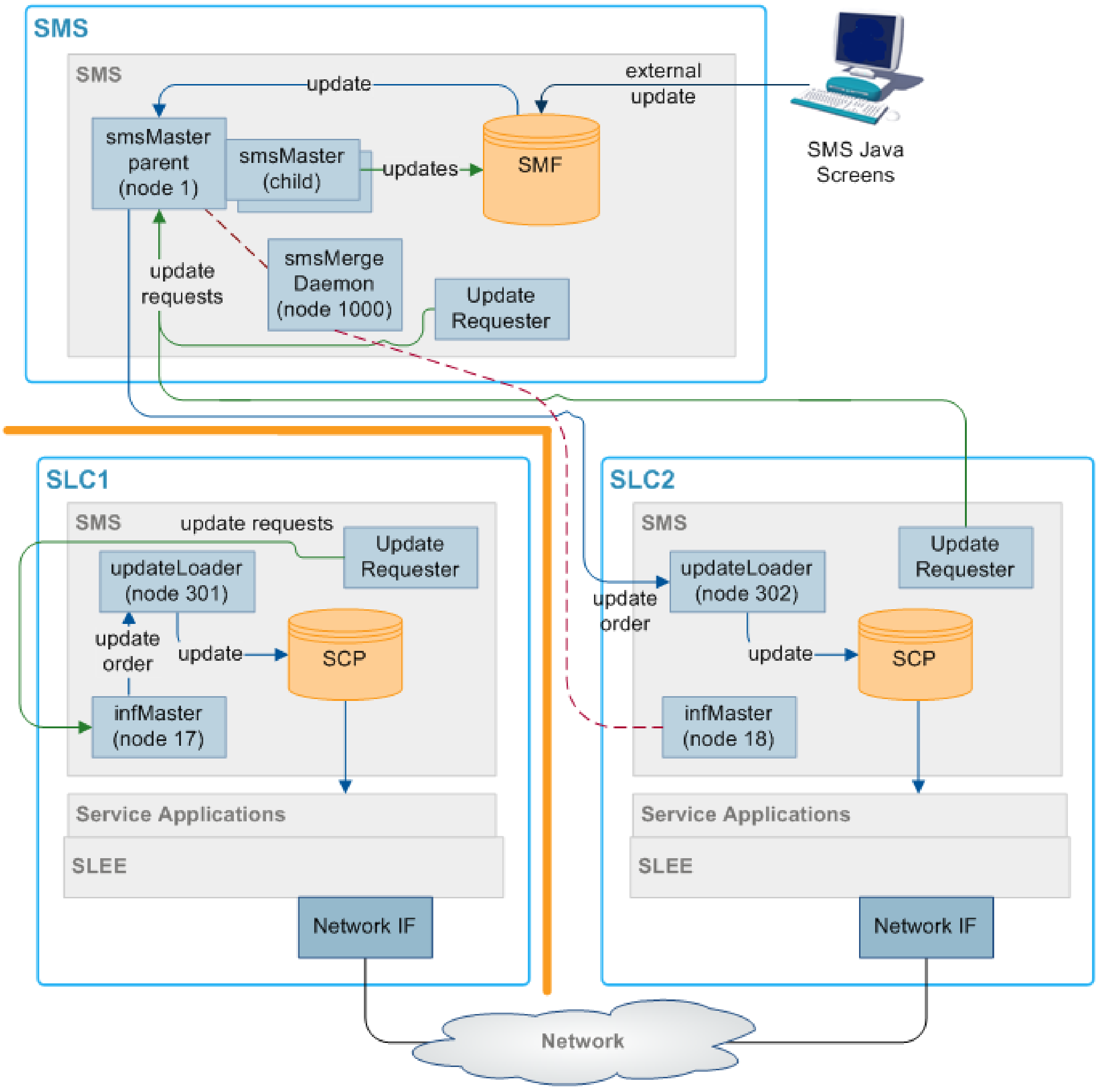
All nodes connected

Here is an example showing all nodes in an unclustered configuration connected to the smsMergeDaemon.



Isolated SLC

This diagram depicts an isolated SLC in an unclustered environment.



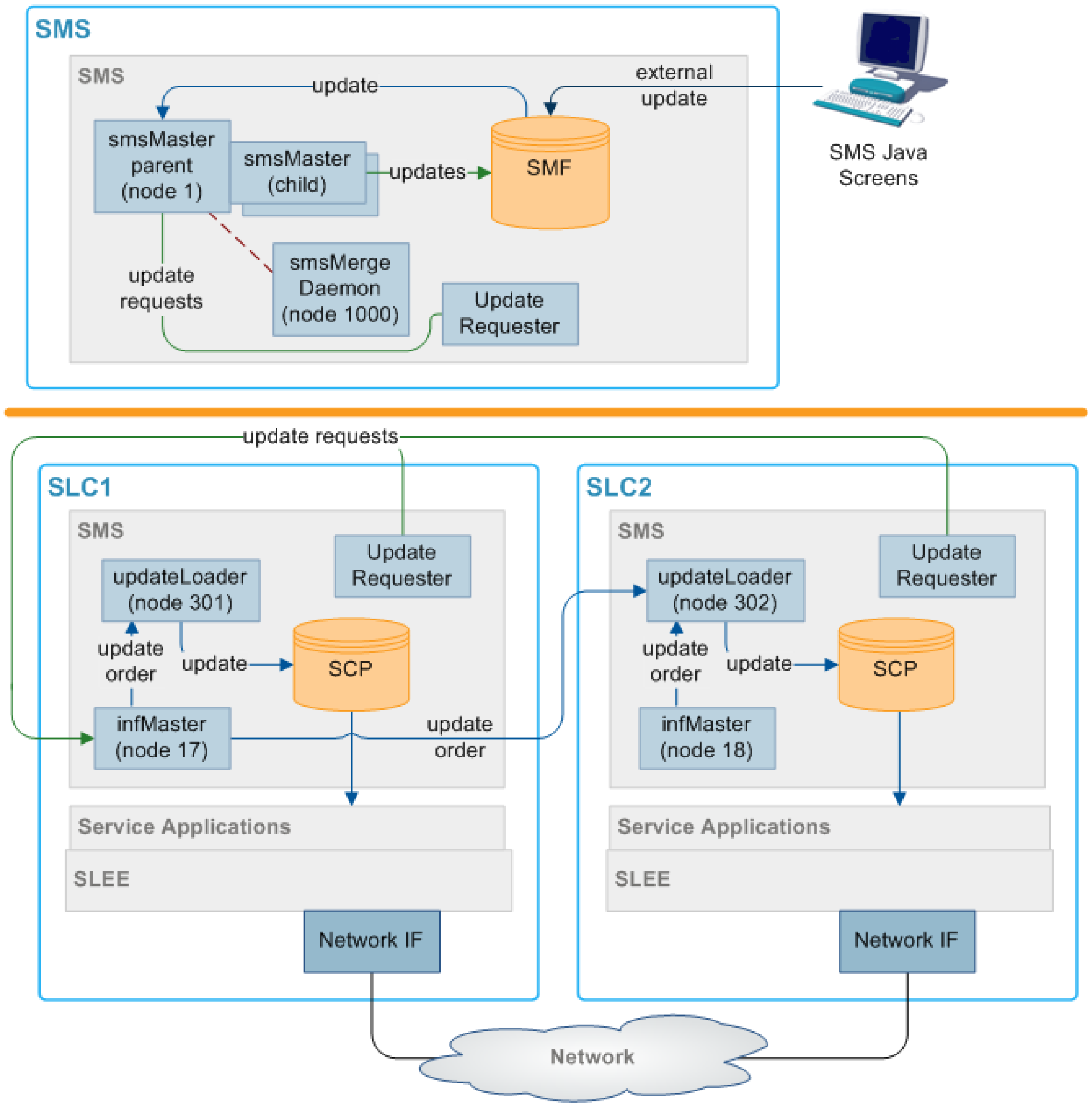
Where an SLC has been isolated from the master it looks for and connects to the master in the network, which has the next lowest node number. In the diagram above, SLC1 has been isolated from the network and the update loader cannot find Master 1, so it looks for the master with the lowest node number it can see (in this case it is Inferior Master 2 on SLC1) and connects to that.

The Master 1 queues all updates for SLC1 until such time as it comes back on line. When SLC1 comes back on line, the smsMergeDaemon queries the infMaster process to see if there are any connections to it. If there are any processes connected to the infMaster, the smsMergeDaemon sends a start merge message to the smsMaster. The smsMaster then updates the rest of the network with the information received from SLC1.

If the smsMergeDaemon is not running, the startMerge process may be used instead. startMerge copies the data from SLC1 to the smsMaster. The smsMaster then updates the rest of the network with the information received from SLC1.

Isolated SMS

This diagram depicts an isolated SMS.

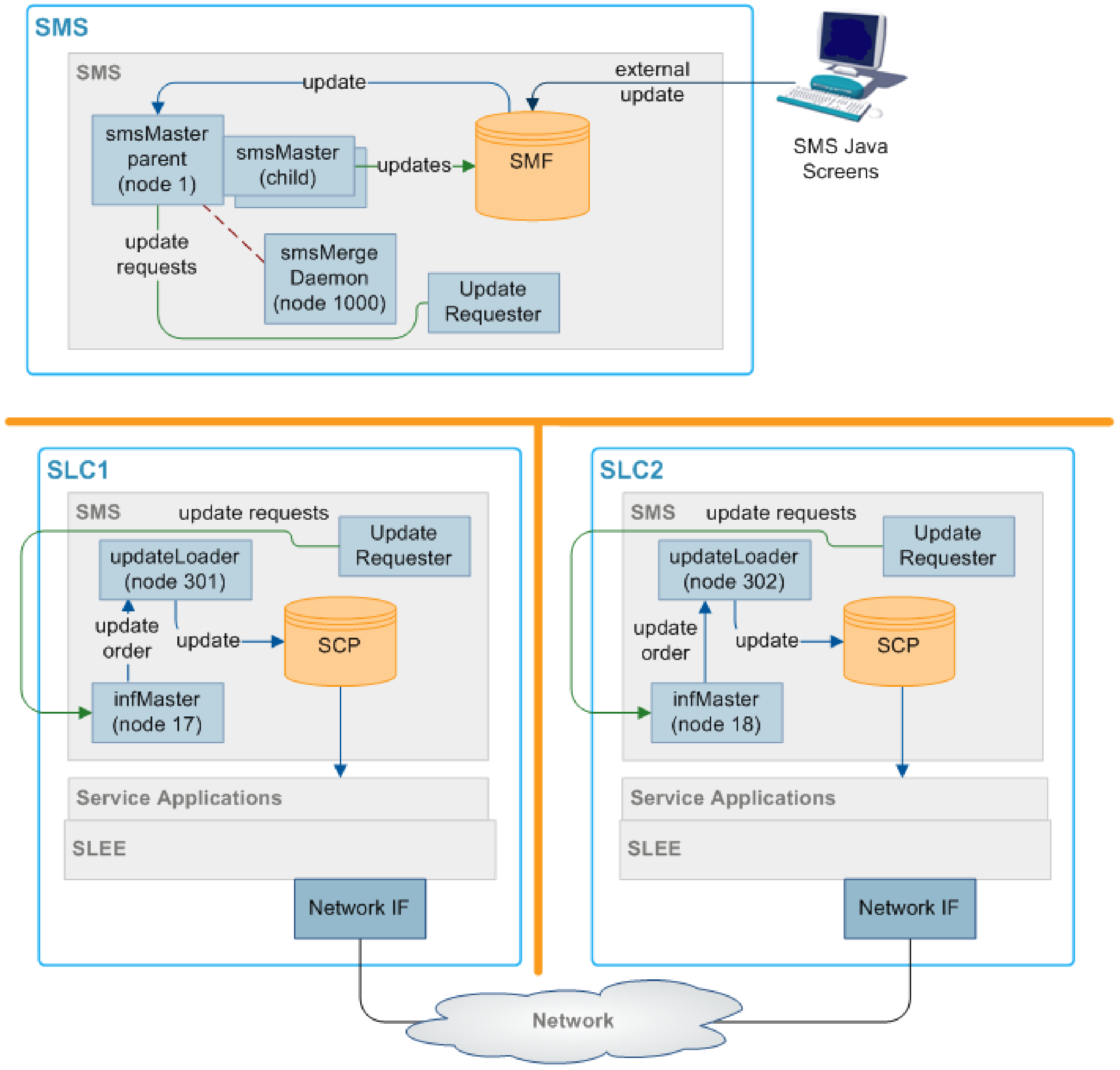


Where the master is isolated from the network, each update loader looks for the inferior master with the lowest node number and connects to that.

In the above case the Master 1 on the SMS has been isolated. The update loader on each node looks for the inferior master with the lowest node number it can find, in this case the update loaders on both SLC1 and SLC2 finds and connecst to inferior master 2 on SLC1. When the SMS comes back into the network, the smsMergeDaemon checks each SLC infMaster process to see if there are any connections to them. In this case, there are connections to the SLC1 infMaster process (node 2) from the SLC2 (node 3). The smsMergeDaemon runs startMerge against SLC1. startMerge copies SLC1's data across to the SMS. The smsMaster then attempts to update both SLCs with the new data from SLC1.

All nodes isolated

This diagram depicts all nodes isolated.



Where all nodes in the network are isolated, they each connect to the inferior master with the lowest node number that they can see. In the above example, this results in the update loader on SLC1 connecting to inferior master 2 on SLC1 (node 2), and the update loader on SLC2 connecting to the inferior master 3 (node 3).

As the SLCs reconnect to the SMS and reestablish a reliable heartbeat, the smsMergeDaemons run startMerge against each SLC to copy the data across to the SMS. Then the SMS replicates the data to the available SLCs.

Merging nodes

If a infMaster is acting as a superior master, it collects update requests in a table on the local SCP. When the smsMaster (or another infMaster with a higher node number) reconnects, all local update requests must be forwarded to the new superior master node and replicated.

The process for completing this task is known as a merge. Usually, the smsMergeDaemon initiates a merge automatically when the connection has stabilized. However, it is also possible to start a merge by hand by invoking the startMerge process from the command line.

For more information about using startMerge, see *startMerge* (on page 207).

Description of resync processes and executables

This table describes the roles of the components involved in the resync process.

Process	Role	Further information
smsMaster	The smsMaster collects update requests in the pending update queue until the destination updateLoader acknowledges a successful update. If the smsMaster cannot connect to a updateLoader, it collects pending updates until a new connection to the updateLoader is made.	*smsMaster* (on page 120)
resyncServer	Takes a snapshot of the SMF and sends it to the compareResyncReceive process on the SLC. One resyncServer is started for each resync commenced.	resyncServer
smsCompareResyncServer	Reads configuration information from the configuration file created by resyncServer and starts a resync.	*smsCompareResyncServer* (on page 184)
compareResyncReceiver	Updates the SCP with the data from the SMF (sent by resyncServer on the SMS).	
smsCompareResyncClient	Receives information from smsCompareResyncServer and updates the SCP.	*smsCompareResyncClient* (on page 181)
updateLoader	When a resync is started, the updateLoader stops making updates to the SCP. Instead it writes the updates to a file named in the following file: *nodenum*-**queuedOrders.da**t When the resync is completed, the queued update orders are processed as normal.	*updateLoader* (on page 168)

replication.def File

Introduction

The **replication.def** file defines default values for all the replication executables on the node it is on. Any of the defaults may be overridden on the command line when the executable is started.

Example: `MAX PENDING=200` can be overridden when starting an smsMaster by adding the command line parameters `-maxpending 400` (no spaces in the parameter and all lower case).

Note: Ensure that the heartbeat settings for both ends of a heartbeat are set to the same value. Otherwise, the connection is repeatedly dropped.

This file is located in the **/IN/service_packages/SMS/etc/** directory.

Parameters - replication.def

The **replication.def** accepts the following configurable parameters.

COMMIT IDLE TIME

Syntax:	`COMMIT IDLE TIME=`*`mseconds`*
Description:	Timeout period (in milliseconds) for the Update Receiver (Update Loader) to become idle after an Update Request (Update Order) and commit.
Type:	Integer
Optionality:	Optional (default used if not set)
Allowed:	
Default:	100
Notes:	
Example:	`COMMIT IDLE TIME=100`

COMMIT BUSY TIME

Syntax:	`COMMIT BUSY TIME=`*`mseconds`*
Description:	Timeout period (in milliseconds) for the Update Receiver or updateLoader to commit a change even if it remains continuously busy.
Type:	Integer
Optionality:	Optional (default used if not set)
Allowed:	
Default:	10000
Notes:	
Example:	`COMMIT BUSY TIME=10000`

CONFIG DIR

Syntax:	`CONFIG DIR=`*`dir`*
Description:	The directory where the replication config file is stored. This parameter has been included for future development, it is recommended that the default is always used.
Type:	String
Optionality:	Optional (default used if not set)
Allowed:	
Default:	/IN/service_packages/SMS/etc
Notes:	
Example:	`CONFIG DIR=/IN/service_packages/SMS/etc`

CONN RETRY TIME

Syntax:	`CONN RETRY TIME=`*`seconds`*
Description:	Time (in seconds) before an updateLoader tries to reconnect to a master replicator if none is available.
Type:	Integer
Optionality:	Optional (default used if not set)
Allowed:	
Default:	0

Notes:	If set to 0, no re-attempt is made.
Example:	`CONN RETRY TIME=0`

CONNECTION TIMEOUT

Syntax:	`CONNECTION TIMEOUT=`*`seconds`*
Description:	Timeout (in seconds) before an attempted connection to a master is terminated and alternative is tried.
Type:	Integer
Optionality:	Optional (default used if not set)
Allowed:	
Default:	1
Notes:	
Example:	`CONNECTION TIMEOUT=1`

HB PERIOD

Syntax:	`HB PERIOD=`*`seconds`*
Description:	Heartbeat period (in seconds)
Type:	Integer
Optionality:	Optional (default used if not set)
Allowed:	
Default:	10
Notes:	The period should be consistent across all platforms. Not advisable to take below 3 seconds.
Example:	`HB PERIOD=10`

HB TIMEOUT

Syntax:	`HB TIMEOUT=`*`seconds`*
Description:	Heartbeat timeout period (in seconds) used by smsMergeDaemon, before a heartbeat is considered late.
Type:	Integer
Optionality:	Optional (default used if not set)
Allowed:	
Default:	10
Notes:	Generally set to the same as HB PERIOD.
Example:	`HB TIMEOUT=10`

HB TOLERANCE

Syntax:	`HB TOLERANCE=`*`mseconds`*
Description:	Heartbeat tolerance time (in millisecs).
Type:	Integer
Optionality:	Optional (default used if not set)
Allowed:	
Default:	250
Notes:	Default is generally used
Example:	`HB TOLERANCE=250`

`HTML DIR`

Syntax:	`HTML DIR=`*`dir`*
Description:	The directory where html files are written.
Type:	String
Optionality:	Optional (default used if not set)
Allowed:	
Default:	/IN/html
Notes:	
Example:	`HTML DIR=/IN/html`

`LONG TIMEOUT`

Syntax:	`LONG TIMEOUT=`*`seconds`*
Description:	Heartbeat timeout period (in seconds) used by smsMergeDaemon to check if the connections to smsMaster and the node to be merged are stable. If they have both been responding to heartbeats within the time specified in `LONG TIMEOUT`, the merge takes place.
Type:	Integer
Optionality:	Optional (default used if not set)
Allowed:	
Default:	60
Notes:	
Example:	`LONG TIMEOUT=60`

`MASTER PORT`

Syntax:	`MASTER PORT=`*`port`*
Description:	The TCP port that master replicators listen for connections on.
Type:	Integer
Optionality:	Optional (default used if not set)
Allowed:	
Default:	12343
Notes:	Generally the default is used
Example:	`MASTER PORT=12343`

`MAXMASTERSNODES`

Syntax:	`MAXMASTERSNODES=`*`num`*
Description:	The number of master nodes used.
Type:	Integer
Optionality:	Optional (default used if not set)
Allowed:	
Default:	8
Notes:	
Example:	`MAXMASTERSNODES=8`

MAX PENDING

Syntax:	`MAX PENDING=`*`num`*
Description:	Used by master replicators to determine maximum size of their pending updates queue (the maximum number of outstanding updates that are stored before an unconnected updateLoader is considered "Out Of History").
Type:	Integer
Optionality:	Optional (default used if not set)
Allowed:	
Default:	10000
Notes:	
Example:	`MAX PENDING=10000`

NODE ID

Syntax:	`NODE ID=`*`ID`*
Description:	Used by an updateLoader to define its replication node number. This value must be unique, and is set to the default value at installation. If more than one updateLoader is running on the same SLC machine, you must override this value with a unique number, for example, by setting the `nodeid` command line parameter.
Type:	Integer
Optionality:	Required
Default:	274
Example:	`NODE ID=274`

ORACLE USER

Syntax:	`ORACLE USER=`*`user/pwd`*
Description:	Oracle username and associated password normally of the form user/password. Operator accounts are used to maintain security.
Type:	String
Optionality:	Optional (default used if not set)
Allowed:	
Default:	/
Notes:	It is recommended that this is left as the default.
Example:	`ORACLE USER=/`

POLLING INTERVAL

Syntax:	`POLLING INTERVAL=`*`useconds`*
Description:	Used to specify the polling interval (in microseconds) when the smsMaster is not receiving replication updates.
Type:	Integer
Optionality:	Optional (default used if not set)
Allowed:	
Default:	50000
Notes:	
Example:	`POLLING INTERVAL=50000`

`QUEUE WARN THRESH`

Syntax:	`QUEUE WARN THRESH=`*`int`*
Description:	The threshold intervals at which warnings are sent to the error log to indicate an increasing or decreasing pending updates queue.
Type:	Integer
Optionality:	Optional (default used if not set)
Allowed:	
Default:	50
Notes:	
Example:	`QUEUE WARN THRESH=50`

`QUEUE ERR THRESH`

Syntax:	`QUEUE ERR THRESH=int`
Description:	The threshold intervals at which a warning is turned into an error and sent to the error log to indicate an increasing/decreasing pending updates queue.
Type:	Integer
Optionality:	Optional (default used if not set)
Allowed:	
Default:	200
Notes:	To work correctly, this must be greater than the QUEUE WARN THRESH value.
Example:	`QUEUE ERR THRESH=200`

`QUEUE CRIT THRESH`

Syntax:	`QUEUE CRIT THRESH=`*`int`*
Description:	The threshold intervals at which a warning or error is turned into a critical error and sent to the error log to indicate an increasing/decreasing pending updates queue.
Type:	Integer
Optionality:	Optional (default used if not set)
Allowed:	
Default:	
Notes:	To work correctly, this must be greater than the QUEUE ERR THRESH value.
Example:	`QUEUE CRIT THRESH=400`

`REP_PATH`

Syntax:	`REP_PATH=`*`path`*
Description:	The directory path of the **replication.config** file.
Type:	String
Optionality:	Optional (default used if not set)
Allowed:	
Default:	**/IN/service_packages/SMS/etc/replication.config**
Notes:	Used by smsMergeDaemon.
Example:	**`REP_PATH=/IN/service_packages/SMS/etc/replication.config`**

`REPORT DIR`

Syntax:	`REPORT DIR=`*`dir`*
Description:	The directory where replication reports (for example, merge reports and database comparison reports) are stored.
Type:	String
Optionality:	Optional (default used if not set)
Allowed:	
Default:	**/IN/service_packages/SMS/output/Replication**
Notes:	
Example:	`REPORT DIR=/IN/service_packages/SMS/output/Replication`

`RESYNC DIR`

Syntax:	`RESYNC DIR=`*`dir`*
Description:	The directory where an updateLoader's **pendingUpdates.dat** file is stored during a resync.
Type:	String
Optionality:	Optional (default used if not set)
Allowed:	
Default:	**IN/service_packages/SMS/tmp**
Notes:	
Example:	`RESYNC DIR=IN/service_packages/SMS/tmp`

`SECONDARY DELAY`

Syntax:	`SECONDARY DELAY=`*`useconds`*
Description:	Initial time (in microseconds) that the primary network has to establish a connection before attempting to connect over the secondary network as well.
Type:	Integer
Optionality:	Optional (default used if not set)
Allowed:	
Default:	100000
Notes:	A value of 0 means both networks are attempted immediately.
Example:	`SECONDARY DELAY=100000`

`SMS_PORT`

Syntax:	`SMS_PORT=`*`port`*
Description:	The SMS port used by the smsMergeDaemon process.
Type:	Integer
Optionality:	Optional (default used if not set)
Allowed:	
Default:	7
Notes:	
Example:	`SMS_PORT=7`

`STATSKEY`

Syntax:	`STATSKEY=`*`key`*
Description:	Shared memory key for updateLoader replication statistics.

Type:	Integer
Optionality:	Optional (default used if not set).
Allowed:	
Default:	270198
Notes:	The default value is recommended. If the default is not used, part of the statistics gathering system (dm_sys) can no longer find the statistics.
Example:	`STATSKEY=270198`

`tcpRxMaxBuf`

Syntax:	`tcpRxMaxBuf = `*`bytes`*
Description:	Sets the receive window size in bytes. This is equivalent to the TCP/IP Tunable Parameter `tcp_recv_hiwat` and is set via calls to setsockopt(). This will be limited by `tcp_max_buf`, which is the maximum transmit/receive buffer size in bytes.
Type:	Integer
Optionality:	Optional (default used if not set)
Allowed:	Integer range from 2,048 to 1,073,741,824
Default:	1,048,576
Notes:	The parameter default is different than the system default of 24,576.
Example:	`tcpRxMaxBuf = 2048`

`tcpTxMaxBuf`

Syntax:	`tcpTxMaxBuf = `*`bytes`*
Description:	Sets the transmit window size in bytes. This is equivalent to the TCP/IP Tunable Parameter `tcp_xmit_hiwat` and is set via calls to setsockopt(). This will be limited by `tcp_max_buf`, which is the maximum transmit/receive buffer size in bytes.
Type:	Integer
Optionality:	Optional (default used if not set)
Allowed:	Ranges from 4,096 to 1,073,741,824
Default:	1,048,576
Notes:	The parameter default is different than the system default of 16,384.
Example:	`tcpTxBuf = 4096`

Example replication.def file

Here is an example **replication.def** file for an SMS platform:

```
MAX PENDING=10000
ORACLE USER=/
HB PERIOD=20
HB TIMEOUT=20
LONG TIMEOUT=60
HB TOLERANCE=10000
CONNECTION TIMEOUT=2
SECONDARY DELAY=100000
CONN RETRY=1
QUEUE WARN THRESH=5
POLLING INTERVAL=50000

CONN RETRY TIME=10
```

```
tcpRxMaxBuf=2048
tcpTxMaxBuf=4096
```

replication.config File

Introduction

The **replication.config** file is a binary configuration file that defines the current specific replication setup. It is a binary representation of the replication setup within the SMF created by the repConfigWrite process.

This file is used by all replication nodes on a machine, and must be:

- The same on each machine
- Accessible by each node

The file is written to the directory specified by the output parameter.

Generating replication.config

This file is usually created by clicking **Create Config File** on the **Table Replication** tab of the Node Management window.

Example replication.config

This text shows an example of a **replication.config** file which has been converted using smsDumpRepConfig.

```
smsDumpRepConfig: File /IN/service_packages/SMS/etc/replication.config
smsDumpRepConfig: (PAD = 0)
smsDumpRepConfig: Short listing. Use -v (verbose) for full listing
--------------------------------------------------------------------------------
smsDumpRepConfig: Table, Column, Group definitions...
--------------------------------------------------------------------------------
TABLE [ACS_CALL_PLAN]
TABLE [ACS_CALL_PLAN_PROFILE]
TABLE [ACS_CALL_PLAN_STRUCTURE]
TABLE [ACS_CLI_CALL_PLAN_ACTIVATION]
TABLE [ACS_CUSTOMER]
TABLE [ACS_CUSTOMER_CLI]
TABLE [ACS_CUSTOMER_SN]
TABLE [ACS_FN_TYPE]
TABLE [ACS_GLOBAL_PROFILE]
TABLE [ACS_LANGUAGE]
TABLE [ACS_NETWORK_KEY]
TABLE [ACS_SN_CALL_PLAN_ACTIVATION]
TABLE [SMF_ALARM_MESSAGE]
TABLE [SMF_STATISTICS]
TABLE [SMF_STATISTICS_DEFN]
--------------------------------------------------------------------------------
smsDumpRepConfig: Replication Groups configured for each node...
--------------------------------------------------------------------------------
NODE NUMBER [1] Prim (192.168.0.173) Sec (0.0.0.0)
NODE NUMBER [301] Prim (192.168.0.163) Sec (0.0.0.0)
    GROUP [ACS_CUSTOMER] [Prim=-1] Min=('+0','',') Max=('+9','','')
    GROUP [ACS_FN_TYPE] [Prim=-1] Min=('+0','',') Max=('+9','','')
    GROUP [ACS_CALL_PLAN_PROFILE] [Prim=-1] Min=('+0','',') Max=('+9','','')
    GROUP [ACS_CALL_PLAN_STRUCTURE] [Prim=-1] Min=('+0','',') Max=('+9','','')
    GROUP [ACS_CALL_PLAN] [Prim=-1] Min=('+0','',') Max=('+9','','')
    GROUP [ACS_CUSTOMER_CLI] [Prim=-1] Min=('+0','',') Max=('+9','','')
    GROUP [ACS_CUSTOMER_SN] [Prim=-1] Min=('+0','',') Max=('+9','','')
    GROUP [ACS_LANGUAGE] [Prim=-1] Min=('+0','',') Max=('+9','','')
    GROUP [SMF_STATISTICS_DEFN] [Prim=-1] Min=('!','!',') Max=('~','~','')
    GROUP [ACS_CLI_CALL_PLAN_ACTIVATION] [Prim=-1] Min=('+0','',') Max=('+9','','')
    GROUP [ACS_GLOBAL_PROFILE] [Prim=-1] Min=('+0','',') Max=('+9','','')
    GROUP [ACS_NETWORK_KEY] [Prim=-1] Min=('+0','',') Max=('+9','','')
    GROUP [ACS_SN_CALL_PLAN_ACTIVATION] [Prim=-1] Min=('+0','',') Max=('+9','','')
NODE NUMBER [302] Prim (192.168.0.178) Sec (0.0.0.0)
    GROUP [ACS_CUSTOMER] [Prim=-1] Min=('+0','',') Max=('+9','','')
    GROUP [ACS_FN_TYPE] [Prim=-1] Min=('+0','',') Max=('+9','','')
    GROUP [ACS_CALL_PLAN_PROFILE] [Prim=-1] Min=('+0','',') Max=('+9','','')
    GROUP [ACS_CALL_PLAN_STRUCTURE] [Prim=-1] Min=('+0','',') Max=('+9','','')
    GROUP [ACS_CALL_PLAN] [Prim=-1] Min=('+0','',') Max=('+9','','')
    GROUP [ACS_CUSTOMER_CLI] [Prim=-1] Min=('+0','',') Max=('+9','','')
    GROUP [ACS_LANGUAGE] [Prim=-1] Min=('+0','',') Max=('+9','','')
    GROUP [SMF_STATISTICS_DEFN] [Prim=-1] Min=('!','!',') Max=('~','~','')
    GROUP [ACS_CLI_CALL_PLAN_ACTIVATION] [Prim=-1] Min=('+0','',') Max=('+9','','')
    GROUP [ACS_GLOBAL_PROFILE] [Prim=-1] Min=('+0','',') Max=('+9','','')
    GROUP [ACS_NETWORK_KEY] [Prim=-1] Min=('+0','',') Max=('+9','','')
    GROUP [ACS_CUSTOMER_SN] [Prim=-1] Min=('+0','',') Max=('+9','','')
```

Further information

For more information, see:

- *replication.config File* (on page 38)
- *smsDumpRepConfig* (on page 194)
- *Service Management System User's Guide*

Chapter 3

Replication Check

Overview

Introduction

This chapter explains replication check and data resychronization processes used in SMS.

In this chapter

This chapter contains the following topics.

Replication Checks

Description

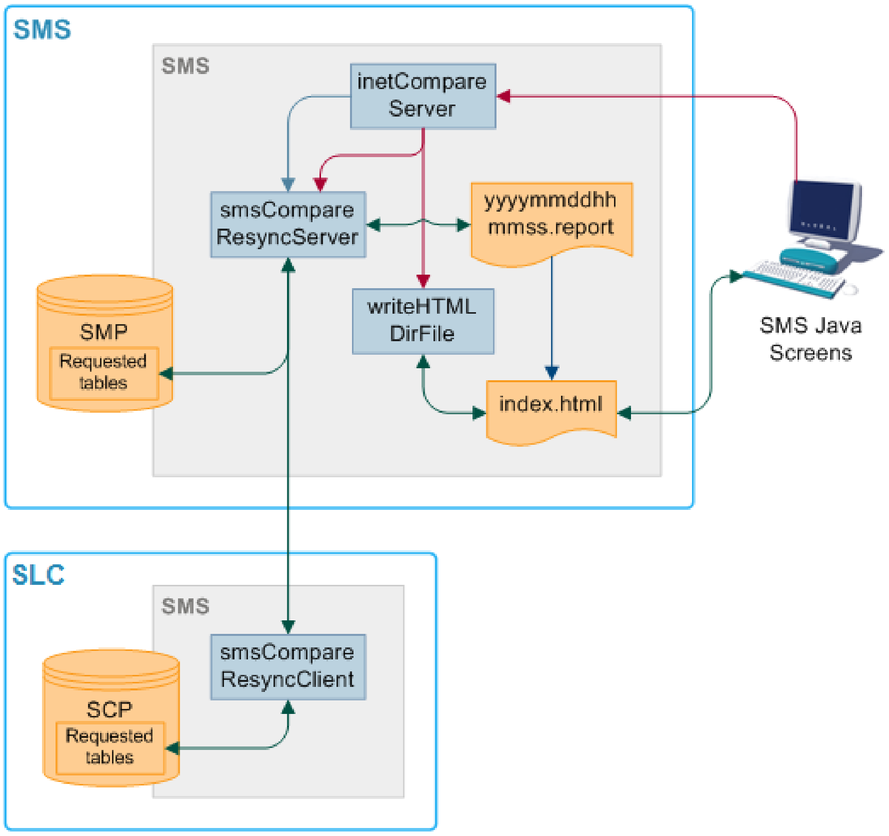
SMS provides a replication check mechanism to enable operators to check the replication of data across their services network.

A replication check will perform a comparison of the SMF data on each replication node. Once the comparison is complete a report will be generated detailing any discrepancies. No data is changed.

Depending on the size of the data set there may be sizable performance impact on the client node and care should be taken to perform such a check outside of peak times.

Replication check diagram

Here is a diagram that shows the elements involved in replication checks.



Replication check components

This table describes the components involved in replication check process.

Process	Role	Further information
SMF	The main database on the SMS.	
SCP	The databases on the SLCs. They hold a subset of the data on the SMF.	
writeHTMLDirFile	Updates **index.html** to include new replication, comparison and resynchronization reports.	
smsCompareResyncServer	Performs database resynchronizations and comparisons on the superior node.	*smsCompareResyncServer* (on page 184)
smsCompareResyncClient	Performs database resynchronizations and comparisons on the inferior node.	*smsCompareResyncClient* (on page 181)
inetCompareServer	Accepts Replication Check requests from the Replication Check screen.	*inetCompareServer* (on page 174)

Replication check process

The replication check process follows these stages.

Stage	Description
1	The `run all` command starts inetCompareServer as configured in the replication check report.
2	inetCompareServer configures and starts *smsCompareResyncServer* (on page 184) and writeHTMLDirFile.
3	*smsCompareResyncClient* (on page 181) handles the other end of the replication check.
4	writeHTMLDirFile updates index.html to include the new report for display in the screens.

Database Comparisons

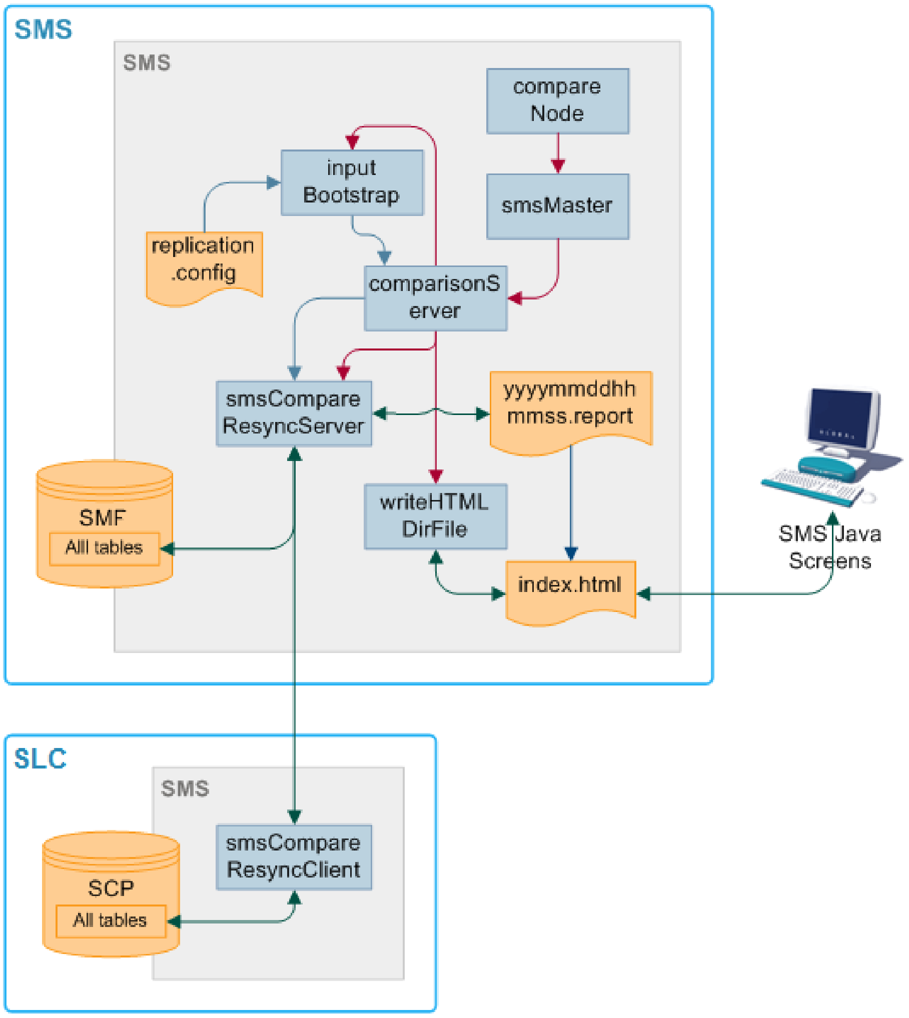
Description

compareNode is used to initiate a full database comparison of an SCP database with the definitive copy in SMF db. This ensures that an SCP's data is consistent with the SMF database. Under normal conditions, this should always be the case, but there may be a time (for example, after multiple failures) where the system administrator wants to check an SLC is consistent.

compareNode tool requests a comparison between the contents of SMF and one other node, by invoking comparisonServer. Comparisons are a more time-efficient method than resyncs. compareServer compares all the entries of all tables which are defined to be replicated to specified updateLoader node.

Database comparison diagram

Here is a diagram that shows the elements involved in a database comparison process.



Database comparison components

This table describes the components involved in database comparison process.

Process	Role	Further information
compareNode	Initiates a full database comparison of an SCP with the definitive copy in SMF. Starts comparisonServer.	*compareNode* (on page 172)
inputBootstrap	Creates configuration files for smsCompareResyncServer from **replication.config**.	*inputBootstrap* (on page 176)
smsMaster	Receives update requests and forwards them to the SMF.	*smsMaster* (on page 120)
comparisonServer	Creates configuration files for smsCompareResyncServer from replication.config.	
smsCompareResync Client	Performs database resynchronizations and comparisons on the inferior node.	*smsCompareResyncClient* (on page 181)

Process	Role	Further information
writeHTMLDirFile	Updates **index.html** to include new replication, comparison and resynchronization reports.	

Database comparison process

The database comparison process follows these stages.

Step	Action
1	compareNode sends a comparison request to smsMaster.
2	smsMaster configures and starts comparisonServer.
3	inputBootstrap provides configuration for comparisonServer from **replication.config**.
4	comparisonServer configures and starts smsCompareResyncServer and writeHTMLDirFile.
5	smsCompareResyncClient handles the other end of the comparison.
6	writeHTMLDirFile updates **index.html** to include the new report for display in the screens.

Database Resynchronizations

Description

Nodes can become out of sync to the point where normal recovery processes cannot rectify the problem. This can happen if there is a network failure for a long period of time, or if there is a fault in the replication process.

If the databases are out of sync, a resynchronization must be run. Resyncs compare the data in two specified nodes and update the inferior database with the different information in the superior database.

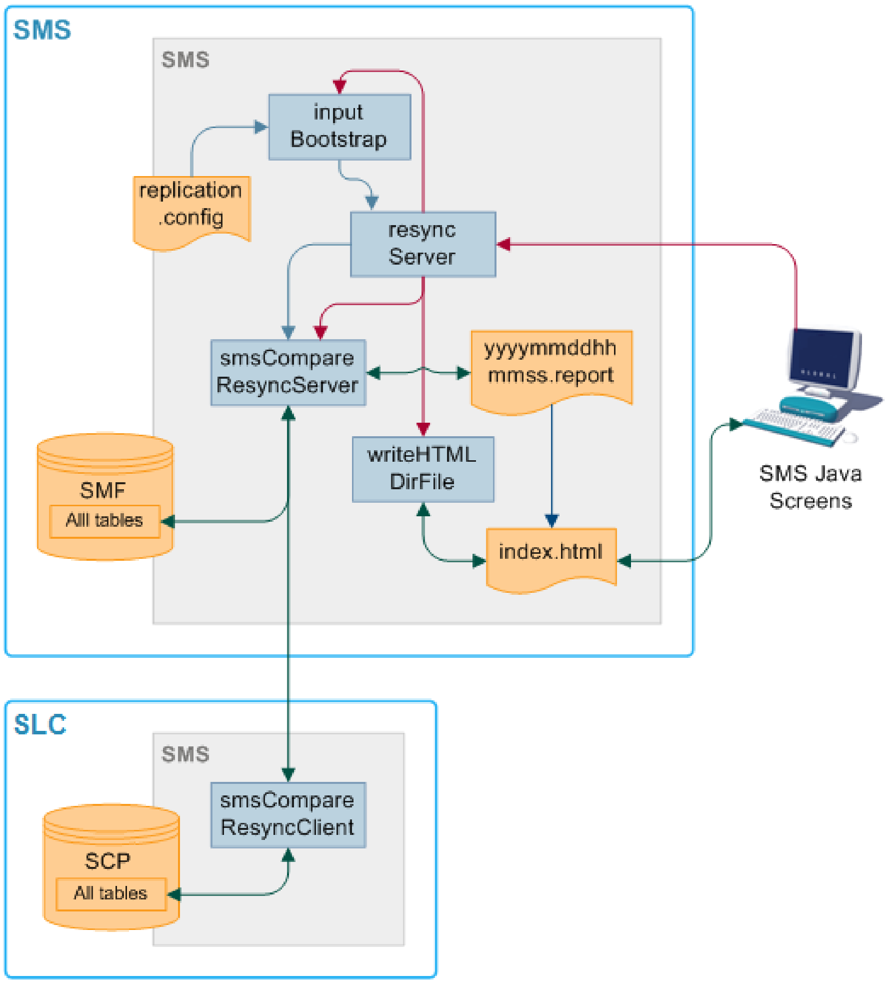
Resyncs run automatically if:

- The updateLoader is started with the -resync flag, and there is a **queuedOrders.dat** file
- The smsMaster has marked that updateLoader node as out of history
- The smsMaster is connected to by an updateLoader which is not in its pending updates queue
- A resync instruction is included in a smsTaskAgent file

Resyncs can also be run from the command line. For more information about running resynchronizations, see *resyncServer* (on page 179).

Database resynchronization diagram

Here is a diagram that shows the elements involved in a database resynchronization process.



Database resynchronization components

This table describes the components involved in database resynchronization process.

Process	Role	Further information
inputBootstrap	Creates configuration files for smsCompareResyncServer from **replication.config**.	*inputBootstrap* (on page 176)
resyncServer	Starts smsCompareResyncServer and inputBootstrap for resyncs.	*resyncServer* (on page 179)
smsCompareResync Client	Performs database resynchronizations and comparisons on the inferior node.	*smsCompareResyncCli ent* (on page 181)
writeHTMLDirFile	Updates **index.html** to include new replication, comparison and resynchronization reports.	

Database resynchronization process

The resynchronization process follows these stages.

Stage	Description
1	If the resync has been started from the command line using resync, resync sends a resynchronization request to smsMaster.
2	smsMaster sends a resynchronization request to resyncServer.
3	resyncServer sends a request to inputBootstrap.
4	inputBootstrap reads data from replication.config and creates a configuration file for smsCompareResyncServer.
5	resyncServer starts smsCompareResyncServer in resync mode.
6	smsCompareResyncServer connects to the SMF and smsCompareResyncClient on the inferior node.
7	smsCompareResyncClient accepts the connection and connects to the SCP.
8	smsCompareResyncServer and smsCompareResyncClient resync the databases.
9	smsCompareResyncServer writes a resynchronization report to **/IN/html/output/SMS/resync/***inferior_node_number***/yyyymmddhhmmss.report**.
10	writeHTMLDirFile updates **/IN/html/output/SMS/resync/***inferior_node_number***/index.html** to include the new report.

Auditing

Description

SMS provides an auditing function for all services implemented through it. It tracks all changes made to the SMF database and stores them in the SMF_AUDIT table. It records:

- User's user ID
- IP address of terminal from which the change was made
- Timestamp of the change
- Table changed
- Copy of the record before the change and a copy of the record after the change

Most database tables also have Change User and Change Date columns which record:

- User name of the user that last changed the contents of a table
- Date at which this change was made

User actions that are logged include:

- Changing a user's password
- Performing a search from the Subscriber tab in the Subscriber Management screen, if only one record is found
- Opening a subscriber record in the Edit Subscriber screen
- Viewing an EDR in the EDR viewer
- Logging in and out of the Customer Care Portal (CCP)
- Locking and unlocking the CCP
- Performing a search in the CCP dashboard

- Viewing a subscriber record through the CCP Quick View

Auditing - listAudit.sh

Audit information is produced as a report by running **listAudit.sh**.

Run either as a cron job or from command line using this command:

```
listAudit.sh usr/pwd [start_date] [end_date] [db_user] [table]
```

Chapter 4

Configuring the Environment

Overview

Introduction

This chapter explains the steps required to configure Oracle Communications Convergent Charging Controller Service Management System (SMS).

In this chapter

This chapter contains the following topics.

Configuration Overview

Introduction

This topic provides a high level overview of how the SMS application is configured. Configuration details for individual processes are located with the documentation for that process.

Configuration process overview

This table describes the steps involved in configuring SMS for the first time.

Stage	Description
1	The environment SMS runs in must be configured correctly. This includes: • If the directory SMS was installed into was not the recommended directory, setting the root directory • If this was a clustered installation, configuring the resource groups • Configuring the Oracle wallet • Configuring the Oracle listener • Configuring the SNMP agent • Configuring connections for CORBA services

Stage	Description
	• Configuring the location of the EDR directories • Configuring the smf_oper profile • Configuring the webserver
2	The replication groups must be configured.
3	If the default language for the SMS Java administration screens need changing, the new default language must be configured.
4	If the default language for the help system for the SMS Java administration screens needs changing, the new default language must be configured.
5	The SMS screen-based configuration must be completed. This includes checking node configuration and statistics configuration.

Configuration components

SMS is configured by the following components:

Component	Locations	Description	Further Information
SMS Java Administration screens	SMS	The SMS screens provide a graphical interface for configuring many parts of SMS including: • Replication • Statistics • Alarm filtering • Reports	*Service Management System User's Guide*
replication.def	All machines with running replication agents	This file specifies the configuration parameters for replication. These parameters may also be specified on the command-line for each application.	*replication.def File* (on page 30)
replication.config	All machines with running replication agents	This file holds a binary version of the configuration held in the SMF. It is copied out to all machines and is required by all replication agent.	*replication.config File* (on page 38)
logjob.conf	All SMSs	This file is automatically generated when the smsSms package is installed.	*logjob.conf* (on page 159)
snmp.cfg	All SMSs	This file configures the SNMP agent's details.	*Configuring the SNMP Agent* (on page 70)
repsib.cfg	All SLCs	This is the template file for the updateRequester program. Other applications typically append template definitions to this file when they are installed, and remove them when they are uninstalled.	

About Configuration for Secure SSL Connection to the Database

Convergent Charging Controller supports secure network logins through Secure Socket Layer (SSL) connections from the Convergent Charging Controller UI to the database. SSL is the default method for connecting to the database when you install Convergent Charging Controller.

To enable SSL connections to the database, the following additional configuration must be set in the **sms.jnlp** file:

- The secureConnectionDatabaseHost Java applet parameter (on non-clustered systems) or the secureConnectionClusterDatabaseHost Java applet parameter (on clustered systems) must specify the database connection in the CONNECT_DATA part. In addition the PROTOCOL part must be set to TCPS and the PORT part must be set to 2484.
- If present, the EncryptedSSLConnection Java applet parameter should be set to true. The Convergent Charging Controller UI connects to the database by using encrypted SSL connections by default.

Note: If you are using non-SSL connections to the database then you must set EncryptedSSLConnection to false. When EncryptedSSLConnection is set to false, the secureConnectionDatabaseHost and the secureConnectionClusterDatabaseHost parameters are ignored.

See *Java Applet Parameters* (on page 86) for more information.

In addition, to enable SSL connections to the database:

- The Oracle wallet that identifies the database server must be created on the SMS node, and its location must be specified in the **listener.ora** and **sqlnet.ora** files. See *Configuring the Oracle Wallet* (on page 56) for more information.
- The **listener.ora** file must be changed to additionally listen on port 2484 by using the TCPS protocol for secure SSL connections to the database. See *Configuring the Oracle Listener* (on page 65) for more information.

Note: The standard Oracle listener TCP port is 1521. However, SSL connections use the standard port for the TCPS protocol, port 2484 instead. If there is a firewall between screen clients and the SMS then you will need to open port 2484 in the firewall.

Configuring the Resource Group in the Clustered Environment

Overview

Certain tasks performed by the cluster require only one instance running across all cluster nodes. For example, an application which modifies a shared data source. There must be a mechanism in place to monitor the running processes and make sure they are restarted when problems arise. This is similar to normal UNIX inittab functionality with the caveat that a process can be restarted on any of the cluster nodes (failover).

Configuration of resource groups must be completed on each node in the cluster.

Starting the webserver failover

Follow these steps to start the httpd failover.

Step	Action
1	Change to the ESERVHttpd directory. **Example command:** `cd /opt/ESERVHttpd`
2	Read the readme file.
3	Stop apache. **Example command:** `/usr/apache/bin/apachectl stop`

Step	Action
4	Change to the util directory within the httpd directory. **Example command:** `cd util/`
5	Start httpd failover using the following command. `startHttpd -h hostname -p "port/tcp"` **Where:** • *hostname* is the shared hostname for the SMS cluster • *port* is the port number the webserver accepts httpd requests on **Example command:** `startHttpd -h smpVirtualCluster -p "80/tcp"` For more information about shared hostnames for clustered machines, see the Oracle documentation.

Starting the sshd failover

Follow these steps to start the sshd failover.

Step	Action
1	Change to the ESERVSshd directory. **Example command:** `cd /opt/ESERVSshd`
2	Read the readme file.
3	Stop the sshd. **Example command:** `/etc/init.d/sshd stop`
4	Change to the util directory in ESERVSshd. **Example command:** `cd util`
5	Start the sshd failover with the following command: `startSshd -h hostname -p "port/tcp"` **Where:** • *hostname* is the shared hostname of the SMS cluster • *port* is the port number the sshd should be running on **Example command:** `startSshd -h smpVirtualCluster -p "22/tcp"` For more information about shared hostnames for clustered machines, see the Oracle documentation.

Starting the smsAlarmDaemon failover

Follow these steps to start the *smsAlarmDaemon* (on page 95) failover.

Step	Action

1	Unset the $HOSTNAME environmental variable.
	Example commands:
	`echo $HOSTNAME`
	`unset HOSTNAME`
	`echo $HOSTNAME`

2	Change to the OracleSmsAlarmDaemon/util directory. **Example command:** `cd /opt/OracleSmsAlarmDaemon/util`

Step	Action
3	Start the smsAlarmDaemon. **Example command:** `./startSmsAlarmDaemon` **Result:** The following information is sent to stdout:

```
Creating a scalable instance ...
Registering resource type <Oracle.SmsAlarmDaemon>...done.
Creating scalable resource group <SmsAlarmDaemon-sarg>...done.
Creating resource <SmsAlarmDaemon-sars> for the resource type
<Oracle.SmsAlarmDaemon>...done.
Bringing resource group <SmsAlarmDaemon-sarg> online...done.
```

Starting the smsAlarmRelay failover

Follow these steps to start the *smsAlarmRelay* (on page 99) failover.

Step	Action
1	Change to the OracleSmsAlarmRelay directory. **Example command:** `cd /opt/OracleSmsAlarmRelay`
2	Read the readme file.
3	Change to the util directory in the OracleSmsAlarmRelay. **Example command:** `cd util`
4	Start the smsAlarmRelay. **Example command:** `./startSmsAlarmRelay` **Result:** The following information is sent to stdout:

```
Creating a failover instance ...
Registering resource type <Oracle.SmsAlarmRelay>...done.
Creating failover resource group <SmsAlarmRelay-harg>...done.
Creating resource <SmsAlarmRelay-hars> for the resource type
<Oracle.SmsAlarmRelay>...done.
Bringing resource group <SmsAlarmRelay-harg> online...done.
```

Starting the smsNamingServer failover

Follow these steps to start the *smsNamingServer* (on page 121) failover.

Step	Action
1	Change to the OracleSmsNamingServer/util directory. **Example command:** `cd /opt/OracleSmsNamingServer/util`
2	Start the smsNamingServer failover. **Example command:** `./startSmsNamingServer` **Result:** The following information is sent to stdout:

```
Creating a scalable instance ...
Registering resource type <Oracle.SmsNamingServer>...done.
Creating scalable resource group <SmsNamingServer-sarg>...done.
Creating resource <SmsNamingServer-sars> for the resource type
<Oracle.SmsNamingServer>...done.
Bringing resource group <SmsNamingServer-sarg> online...done.
```

Starting the smsReportScheduler failover

Follow these steps to start the *smsReportScheduler* (on page 123) failover.

Step	Action
1	Change to the OracleSmsReportScheduler/util directory. **Example command:** `cd /opt/OracleSmsReportScheduler/util`
2	Start the smsReportScheduler failover. **Example command:** `./startSmsReportScheduler` **Result:** The following information is sent to stdout:

```
Creating a failover instance ...
Registering resource type <Oracle.SmsReportScheduler>...done.
Creating failover resource group <SmsReportScheduler-harg>...done.
Creating resource <SmsReportScheduler-hars> for the resource type
<Oracle.SmsReportScheduler>...done.
Bringing resource group <SmsReportScheduler-harg> online...done.
```

Starting the smsReportsDaemon failover

Follow these steps to start the *smsReportsDaemon* (on page 122) failover.

Step	Action
1	Change to the OracleSmsReportsDaemon/util directory. **Example command:** `cd /opt/OracleSmsReportsDaemon/util`
2	Start the smsReportsDaemon failover. **Example command:** `./startSmsReportsDaemon` **Result:** The following information is sent to stdout:

```
Creating a scalable instance ...
Registering resource type <Oracle.SmsReportsDaemon>...done.
Creating scalable resource group <SmsReportsDaemon-sarg>...done.
Creating resource <SmsReportsDaemon-sars> for the resource type
<Oracle.SmsReportsDaemon>...done.
Bringing resource group <SmsReportsDaemon-sarg> online...done.
```

Starting the smsStatsThreshold failover

Follow these steps to start the *smsStatsThreshold* (on page 137) failover.

Step	Action
1	Change to the OracleSmsStatsThreshold/util directory. **Example command:** `cd /opt/OracleSmsStatsThreshold/util`
2	Start the smsStatsThreshold failover. **Example command:** `./startSmsStatsThreshold` **Result:** The following information is sent to stdout:

```
Creating a failover instance ...
Registering resource type <Oracle.SmsStatsThreshold>...done.
Creating failover resource group <SmsStatsThreshold-harg>...done.
Creating resource <SmsStatsThreshold-hars> for the resource type
<Oracle.SmsStatsThreshold>...done.
Bringing resource group <SmsStatsThreshold-harg> online...done.
```

Starting the smsTaskAgent failover

Follow these steps to start the *smsTaskAgent* (on page 139) failover.

Step	Action
1	Change to the OracleSmsTaskAgent/util directory. **Example command:** `cd /opt/OracleSmsTaskAgent/util`

2	Start the smsTaskAgent.
	Example command: `./startSmsTaskAgent`
	Result: The following information is sent to stdout:
	`Creating a scalable instance ...`
	`Registering resource type <Oracle.SmsTaskAgent>...done.`
	`Creating scalable resource group <SmsTaskAgent-sarg>...done.`
	`Creating resource <SmsTaskAgent-sars> for the resource type`
	`<Oracle.SmsTaskAgent>...done.`
	`Bringing resource group <SmsTaskAgent-sarg> online...done.`

Configuring Replication Files

Introduction

There are two configuration files for replication that may be changed by the administrator.

- **replication.def**
- **replication.config**

The replication.config file

The **replication.config** file is created and changed through the Node Management screens in the SMS screens. The user must move tables on the screen from the Available Replication Groups list to the node they are to be replicated to in the Allocated Replication Groups list. Clicking **Create Config File** produces a new **replication.config** file.

The previous configuration is deleted prior to the new configuration being loaded. This does not necessitate the application being restarted, but it causes disruption to service on any of the SLCs.

The **replication.config** file contains the configuration for the whole network. This includes all configuration details needed for smsMasters and infMasters (if necessary).

Implementing changes to the replication.config file

The new **replication.config** file takes effect after the program called changeConfig is run.

If you make the new configuration from the screens, this will be immediately. If you make the config from the command line, the change can be scheduled.

The replication.def file

The **replication.def** file is configured when the application is installed and should not need to be updated. It contains parameters that may be changed by the operator on start-up.

Implementing change to replication.def

Since **replication.def** is read only when the application starts up, if it does need to be updated and changes are made, the application (updateLoader, infMaster or smsMaster) must be restarted for these changes to take effect. After restart, these changes take effect immediately.

The **Replication.def** file is held on each node in the same directory as the application (updateLoader, infMaster or smsMaster). If changes are made to the SLC configuration, infMaster and updateLoader must be restarted.

Where changes are to be made to the SMS configuration, the smsMasters must be restarted. In an unclustered installation, smsMaster must be shut down by merging it with an infMaster to avoid loss of data and update information.

Example replication.def file

Here is an example of the default **replication.def** file that is installed when you install Convergent Charging Controller:

```
# @(#)replication.def   1.2
MAX PENDING=10000
ORACLE USER=/
HB PERIOD=10
HB TIMEOUT=10
HB TOLERENCE=250
CONNECTION TIMEOUT=2
 SECONDARY DELAY=100000
CONN RETRY=1
QUEUE WARN THRESH=5
#Some Update Loader values
CONN RETRY TIME=0
RESYNC DIR=/IN/service_packages/SMS/tmp
CONFIG DIR=/IN/service_packages/SMS/etc
HTML DIR=/IN/html
REPORT DIR=/IN/service_packages/SMS/output/Replication
```

Configuring the Oracle Wallet

About the Oracle Wallet

The Oracle wallet is the single-sign-on wallet that is used when connecting securely to the database and that contains certificate information for identifying the Oracle server. You must create the Oracle wallet if you are using secure SSL connections to the database.

The certificate identifying the server must be signed by a certificate authority (CA) either by creating a root CA and self-signing, or by sending a certificate signing request to a commercial CA.

You can create the Oracle wallet and server certificate in the following ways:

- Manually by using the Oracle PKI tool, orapki. The orapki tool provides a uniform interface for manipulating Oracle wallets and certificates. See *Manually Creating the Oracle Wallet* (on page 57) for more information.
- Automatically by using the **setupOracleWallet.sh** script. This script automatically issues the orapki commands and prompts you for the required information. See *Creating the Oracle Wallet Automatically by Using setupOracleWallet.sh* (on page 62) for more information.

On a clustered SMS you should create the Oracle wallet in a file system that is cluster-wide to allow all instances to access the same wallet information in a single location; for example, on a non-clustered SMS node the Oracle wallet is located in the following directory by default:

/u01/app/wallets/oracle/server

However, if **/global** is a shared volume on a cluster then you should use the following directory for the Oracle wallet:

/global/oracle/app/wallets/oracle/server

About Configuring the Location of the Oracle Wallet

The Oracle wallet is used for single sign on to the Oracle server. If you are using secure SSL connections to the database then you must configure the location of the Oracle wallet in the WALLET_LOCATION entry in the **listener.ora** and **sqlnet.ora** files by using the following syntax:

```
WALLET_LOCATION =
     (SOURCE =
          (METHOD = FILE)
          (METHOD_DATA =
          (DIRECTORY = directory))
)
```

Where *directory* is the directory where the Oracle auto-login wallet is located; for example, on a non-clustered system the Oracle wallet default location is:

/u01/app/wallets/oracle/server

On a clustered system, the Oracle wallet default location is:

/global/oracle/app/wallets/oracle/server

Note: On a clustered system you should specify a cluster-wide shared location so that a single Oracle wallet definition can be accessed from all cluster nodes.

In addition you must configure the following entries in the **listener.ora** and the **sqlnet.ora** files:

```
SSL_CLIENT_AUTHENTICATION=FALSE
SSL_CIPHER_SUITES = (TLS_RSA_WITH_AES_128_CBC_SHA)
```

You must also set the sslCipherSuites Java applet parameter in the **sms.jnlp** file to the same value as the SSL_CIPHER_SUITES entry.

Manually Creating the Oracle Wallet

The following high-level procedure explains how to create the Oracle wallet by using the Oracle orapki tool, and how to add a trusted or a self-signed certificate to the server wallet.

Follow these steps to create the Oracle wallet and add a trusted or a self-signed certificate to the server wallet.

Step	Action
1	(Optional) Skip this step if you are using a commercial CA to sign the server certificate. If you want to use self-signed certificates, then you must create the wallet container for the root CA. See *Creating the Wallet Container for the Root CA* (on page 58) for more information.
2	(Optional) Skip this step if you are using a commercial CA to sign the server certificate. If you want to use self-signed certificates, then you must create a self-signed certificate. See *Creating a Self-Signed Certificate* (on page 59) for more information.
3	Create an Oracle wallet to store the Oracle server certificate. See *Creating an Oracle Wallet to Store the Oracle Server Certificate* (on page 59) for more information.
4	Add a user certificate to the server wallet. See *Adding a User Certificate to the Server Wallet* (on page 60) for more information.

Step	Action
5	Export a certificate-signing request from the server wallet. See *Exporting the Server Certificate Request* (on page 60) for more information.
6	Sign the server certificate request. If you are using: • Self-signed certificates, see *Signing the Server Certificate Request by Using the Self-Signed Certificate from the Root CA* (on page 60) for more information. • A commercial CA, see *Signing the Server Certificate Request by Using a Commercial CA* (on page 61) for more information.
7	Configure the pathname to the server wallet in the WALLET_LOCATION entry in the **listener.ora** and **sqlnet.ora** files. See *About Configuring the Location of the Oracle Wallet* (on page 57) for more information.
8	(Optional) Skip this step if you are using a commercial CA. Add the trusted certificates to the keystore on client PCs. See *Adding Trusted Certificates to the Keystore on Client PCs* (on page 62) for more information.

Creating the Wallet Container for the Root CA

This procedure assumes that Convergent Charging Controller is installed on a non-clustered SMS node and that the following directory has been created for the Oracle wallet:

/u01/app/wallets/oracle

On a clustered SMS the Oracle wallet is located in a file system that is cluster-wide to allow all instances to access the same wallet information in a single location; for example, **/global/oracle/app/wallets/oracle.**

Follow these steps to create the wallet container for the root CA.

Step	Action
1	Log in to the SMS as user *oracle*.
2	Go to the directory created for the Oracle wallet, for example: `cd /u01/app/wallets/oracle`
3	Create the wallet container by entering the following command: `orapki wallet create -wallet ./root`
4	When prompted, specify a new password for the root wallet. **Note:** Wallet passwords have length and content validity checks applied to them. Generally passwords should have a minimum length of eight characters and contain alphabetic characters combined with numbers and special characters.
5	Confirm the password. orapki creates the following directory for the root wallet and adds the **ewallet.p12** file in root directory: **/u01/app/wallets/oracle/root**

Creating a Self-Signed Certificate

Follow these steps to create a self-signed certificate in the root wallet and export it to a file named **b64certificate.txt**.

Step	Action
1	Create a self-signed certificate that is added to the root wallet by entering the following command: `orapki wallet add -wallet ./root -dn CN=root_CA,C=CC -keysize 2048 -self_signed -validity 3650` Where root_CA is the self-signed certificate name and *CC* is the local international country code.
2	When prompted, enter the password for the root wallet.
3	Export the self-signed certificate from the root wallet by entering the following command: `orapki wallet export -wallet ./root -dn CN=root_CA,C=CC -cert ./root/b64certificate.txt` Where *CC* is the local international country code, and the `-cert` command line option specifies the location of the export certificate.
4	When prompted, enter the password for the root wallet. The self-signed certificate is exported to the file **b64certificate.txt**.

Creating an Oracle Wallet to Store the Oracle Server Certificate

You create an Oracle wallet in the server sub-directory of the wallet directory to store the Oracle server certificate. The server sub-directory is in addition to the root sub-directory that you optionally created for the root CA.

The server wallet is used to authenticate the Oracle server. The location of the Oracle server wallet must be specified in the following WALLET_LOCATION configuration in **listener.ora** and **sqlnet.ora** files:

```
WALLET_LOCATION =
     (SOURCE =
     (METHOD = FILE)
     (METHOD_DATA = (DIRECTORY = server_directory)
     )
```

Where *server_directory* is the directory you create for the Oracle server certificate; for example:

/u01/app/wallets/oracle/server

See *About Configuring the Location of the Oracle Wallet* (on page 57) for more information.

Follow these steps to create an Oracle wallet for the server certificate.

Step	Action
1	As user `oracle` on the SMS node, go to the Oracle wallet directory; for example: `cd /u01/app/wallets/oracle`

Step	Action
2	Create the server wallet by entering the following command: `orapki wallet create -wallet ./server -auto_login`
3	When prompted specify a new password for the server wallet. **Note:** Wallet passwords have length and content validity checks applied to them. Generally passwords should have a minimum length of eight characters and contain alphabetic characters combined with numbers and special characters.
4	Confirm the password. orapki creates the **/u01/app/wallets/oracle/server** directory for the server wallet and adds the following files in the directory: **cwallet.sso** **ewallet.p12**
5	To check that the files have been created, enter the following command: `ls server`

Adding a User Certificate to the Server Wallet

Follow these steps to add a user certificate for the SMS to the server wallet.

Step	Action
1	As user `oracle` on the SMS, enter the following command to add a user certificate for the SMS to the server wallet: `orapki wallet add -wallet ./server/ewallet.p12 -dn 'CN=SMS,C=CC' -keysize 2048` Where **ewallet.p12** is the name of the server wallet and *CC* is the local international country code.
2	When prompted, enter the password for the server wallet.

Exporting the Server Certificate Request

You export a certificate request from the server wallet so that the request can be signed by a CA.

To export the server certificate request enter the following command as user *oracle*:

```
orapki wallet export -wallet ./server -dn 'CN=SMS,C=CC' -request ./server/creq.txt
```

Where *CC* is the local international country code and **creq.txt** is the name of the server certificate request file.

The server request is exported to the following file in the server directory:

/u01/app/wallets/oracle/server/creq.txt

Signing the Server Certificate Request by Using the Self-Signed Certificate from the Root CA

The following procedure uses the root CA you initially created to sign the certificate request. Alternatively you can send the request to a commercial CA for signing.

Follow these steps to sign the server certificate request.

Step	Action
1	Create the server certificate in the file named **cert.txt** using the certificate request in the file named **creq.txt**. As user `oracle` on the SMS, enter the following command: `orapki cert create -wallet ./root -request ./server/creq.txt -cert ./server/cert.txt -validity 3650` Where the command line option: • `-wallet` specifies to use the self-signed certificate in the root CA to sign the server request • `-cert` specifies to create the signed certificate named **cert.txt**
2	When prompted, enter the password for the root wallet.
3	Add the trusted certificate of the root CA, **./root/b64certificate.txt**, and the user certificate signed by the root CA, **./server/cert.txt**, into the server wallet by entering the following commands: • `orapki wallet add -wallet ./server/ewallet.p12 -trusted_cert –cert ./root/b64certificate.txt` • `orapki wallet add -wallet ./server/ewallet.p12 -user_cert –cert ./server/cert.txt` When prompted, enter the password for the server wallet.

Signing the Server Certificate Request by Using a Commercial CA

Follow these steps to use a commercial CA to sign the server certificate request.

Step	Action
1	Send the certificate request in the file named **creq.txt** to the commercial CA for signing.
2	When you receive the signed certificate back from the commercial CA, add the commercial CA's trusted public certificate to the server wallet container. `orapki wallet add -wallet ./server/ewallet.p12 -trusted_cert –cert` *`trusted_CA_certificate`* Where *trusted_CA_certificate* is the file containing the CA's trusted public certificate.
3	When prompted for a password, enter the password for the server wallet.
4	Add the CA-signed server certificate to the server wallet container. `orapki wallet add -wallet ./server/ewallet.p12 -user_cert -cert` *`CA_signed_certificate`* Where *CA_signed_certificate* is the signed server certificate from the CA.
5	When prompted, enter the password for the server wallet.

Adding Trusted Certificates to the Keystore on Client PCs

If you are using self-signed certificates then you must update the keystore on client PCs to trust certificates from the SMS server that have been signed by the root CA.

Note: Certificates signed by a commercial CA are already trusted by definition, therefore update the keystore on client PCs only if you are using self-signed certificates.

Follow these steps to add a trusted certificate for the SMS server to the Java keystore on a client PC.

Step	Action
1	Copy the root CA certificate **./root/b64certificate.txt** to the client PC.
2	As the `Administrator` user on the client PC, open the command tool window and enter the following command: `keytool -importcert -keystore "path_to_java_lib_security_cacerts" -alias SMS -file "path_to_b64certificate_txt"` where: • *path_to_java_lib_security_cacerts* is the path for the **cacerts** file • *path_to_b64certificate_txt* is the path for the **b64certificate.txt** file
3	When prompted, enter the password for the keystore. **Note:** The Java installation sets the keystore password to **`changeit`** by default.
4	Answer yes to the following prompt: `Trust this certificate? [no]:` Oracle keytool updates the keystore on the client PC to trust certificates from the SMS server that have been signed with the root CA.

Creating the Oracle Wallet Automatically by Using setupOracleWallet.sh

About Creating the Oracle Wallet by Using setupOracleWallet.sh

The Oracle wallet is the single-sign-on wallet that is used when connecting securely to the database and that contains certificate information for identifying the Oracle server. You must create the Oracle wallet if you are using secure SSL connections to the database. See *About the Oracle Wallet* (on page 56) for more information.

The **setupOracleWallet.sh** script enables you to automatically run the orapki commands for creating the Oracle wallet. The script prompts you to enter all the information it requires to create the Oracle wallet. See *setupOracleWallet.sh* (on page 179) for more information about **setupOracleWallet.sh**.

When you run **setupOracleWallet.sh**, you specify whether or not you want to use self-signed certificates. If you are using:

- Self-signed certificates, the script completes after creating the Oracle wallet and self-signed certificate. You must then update the Java keystore on client PCs with the trusted certificates. See *Adding Trusted Certificates to the Keystore on Client PCs* (on page 62) for more information.
- Certificates signed by a commercial CA, the script initially completes after creating the certificate signing request. You must send the certificate signing request to the commercial CA for signing. When the commercial CA returns the signed certificate, you re-run **setupOracleWallet.sh** to add the trusted CA certificate and the signed CA certificate to the Oracle server wallet.

After creating the Oracle wallet, the script prints details of the additional configuration that must be set in the Oracle **listener.ora** and **sqlnet.ora** files. See the discussion on *Configuring the Oracle Listener* (on page 65) for more information.

Information Required by setupOracleWallet.sh

The following table lists the information that is required by the **setupOracleWallet.sh** script.

Required Item	Description
Oracle wallet base directory	The base directory for the Oracle wallet. Specify the base directory to use for the Oracle root and Oracle server wallets. On a clustered SMS specify a file system that is cluster-wide to allow all instances to access the same wallet information in a single location. On a non-clustered system the default location for the Oracle wallet base directory is: **/u01/app/wallets/oracle/** On a clustered system the default location for the Oracle wallet base directory is: **/global/oracle/app/wallets/oracle/**
ISO country code	The local international country (ISO) code for your country. Specify the two-letter code.
Wallet passwords	The password to use for the root CA wallet and the password to use for the server wallet. You will be prompted for the password each time the wallet is accessed. **Note:** Wallet passwords have length and content validity checks applied to them. Generally passwords should have a minimum length of eight characters and contain alphabetic characters combined with numbers and special characters.

Setting Up the Oracle Wallet to Use Self-Signed Certificates by Using setupOracleWallet.sh

Follow these steps to set up the Oracle server wallet to use self-signed certificates by using **setupOracleWallet.sh**.

Step	Action
1	Log in to the SMS as user `oracle`.
2	Enter the following command: **`/IN/service_packages/SMS/bin/setupOracleWallet.sh`**
3	Answer `y` to the following prompt: `Do you wish to proceed with the configuration (y/n):`

Step	Action
4	Enter the following information as required: • The base directory for the Oracle wallet. Specify the base directory to use for the Oracle root and Oracle server wallets. On a clustered SMS specify a file system that is cluster-wide to allow all instances to access the same wallet information in a single location. • The local international country (ISO) code for your country. Specify the two-letter code. • The password to use for the root CA wallet and the password to use for the server wallet. You will be prompted for the password each time the wallet is accessed. **Note:** Wallet passwords have length and content validity checks applied to them. Generally passwords should have a minimum length of eight characters and contain alphabetic characters combined with numbers and special characters.
5	Answer `y` to the following prompt: `Would you like to use a self-signed root certificate to sign the SMS server certificate?` When processing completes, the self-signed root certificate is exported to the following file: **. /root/b64certificate.txt** Where **./root** is a sub-directory of the base directory for the Oracle wallet. You must import this certificate into the Java **lib\security\cacerts** file on each client PC by using the Java keytool utility. See *Adding Trusted Certificates to the Keystore on Client PCs* (on page 62) for more information.

Setting Up the Oracle Wallet to Use CA-Signed Certificates by Using setupOracleWallet.sh

Note: This procedure assumes that the commercial CA's own root certificate is available in the following file:

./root/b64certificate.txt

Where **./root** is a sub-directory of the base directory for the Oracle wallet.

Follow these steps to set up the Oracle server wallet to use certificates signed by a commercial CA by using **setupOracleWallet.sh**.

Step	Action
1	Log in to the SMS as user *oracle*.
2	Enter the following command: **`/IN/service_packages/SMS/bin/setupOracleWallet.sh`**
3	Answer `y` to the following prompt: `Do you wish to proceed with the configuration (y/n):`

Step	Action
4	Enter the following information as required: • The base directory for the Oracle wallet. Specify the base directory to use for the Oracle root and Oracle server wallets. On a clustered SMS specify a file system that is cluster-wide to allow all instances to access the same wallet information in a single location. • The local international country (ISO) code for your country. Specify the two-letter code. • The password the password to use for the server wallet. You will be prompted for the password each time the wallet is accessed. **Note:** Wallet passwords have length and content validity checks applied to them. Generally passwords should have a minimum length of eight characters and contain alphabetic characters combined with numbers and special characters.
5	Answer `n` to the following prompt: `Would you like to use a self-signed root certificate to sign the SMS server certificate?` The script creates the server auto-login wallet and exports the certificate signing request to the following file: **. /server/creq.txt** Where **./server** is a sub-directory of the base directory for the Oracle wallet.
6	Send the certificate signing request to the commercial CA for signing. The commercial CA returns the signed certificate.
7	Place the signed certificate in the following file: **./server/cert.txt**
8	Place the root certificate from the commercial CA in the following file: **./root/b64certificate.txt**
9	Log in as user `oracle` on the SMS and enter the following command: `/IN/service_packages/SMS/bin/setupOracleWallet.sh -s ./server/cert.txt -t ./root/b64certificate.txt -w` *`wallet_base_directory`* Where: • `-s` **`./server/cert.txt`** specifies the location of the signed server certificate • `-t` **`./root/b64certificate.txt`** specifies the location of the root certificate from the commercial CA • `-w` *wallet_base_directory* specifies the Oracle wallet base directory The **setupOracleWallet.sh** script completes by adding the trusted CA certificate and the CA-signed certificate to the server wallet.

Configuring the Oracle Listener

Introduction

In order for the database on the SMS node to operate correctly it requires an Oracle listener. The Oracle listener listens for external requests to connect to a database on the SMS node.

The Oracle listener configuration in this section is defined in the **listener.ora** file on the SMS platform only; specific additional configuration is not required on any of the SLC nodes. This is because the **listener.ora** file on the SLC nodes is part of the standard Oracle installation and should not be changed.

The following high-level procedure explains how to add support to the **listener.ora** file to enable access to Oracle database instances by using the TCPS network protocol for secure SSL connections, or by using the TCP network protocol for non-SSL connections. It does not explain how to create a **listener.ora** file. The process of adding support for TCPS or TCP is also described in the Oracle documentation, however it is outlined here for quick reference.

The task of creating or updating the Oracle listener should be performed by your database administrator. See Chapter 5 (Using Sql*Net) in *Understanding Sql*Net*, which is shipped with Oracle 7 for more information about creating an Oracle listener file.

Note: This is not a comprehensive guide to configuring Oracle Database. Configuring and maintaining a database is a non-trivial task, and if you are unsure how to proceed please consult your database administrator.

Procedure

Follow these steps to configure the Oracle listener.

Step	Action
1	Log in to the SMS as user *oracle*, or enter the following command from a root login to become the user *oracle*: `su - oracle` **Note:** Logging in as the user *oracle* ensures that the path to all the Oracle binaries is correct and that file ownership for Oracle files is preserved.
2	Go to the directory containing the **listener.ora** file. The location of the **listener.ora** file depends on the version of Oracle Database installed and the options selected at installation. It is located in one of the following directories by default: • **$ORACLE_HOME/network/admin** • **/var/opt/oracle/**
3	Edit the **listener.ora** file by using a text editor such as vi; for example: `vi listener.ora`

Step	Action
4	Add ADDRESS entries to ADDRESS_LIST to define the SMS hostname, protocols, and ports to use for connecting to the database. Use the following syntax:

```
LISTENER=
  (DESCRIPTION_LIST =
  (DESCRIPTION=(ADDRESS_LIST=
     (ADDRESS=
         (PROTOCOL=protocol)
         (HOST=hostname)
         (PORT=port_number)
         )))

)
```

where:

- *protocol* is the protocol to use for connecting to the SMF database. You must specify **TCPS** for secure SSL connections, or **TCP** for non-SSL connections
- *hostname* is the hostname of the SMS node
- *port_number* is the number of the port on which the listener listens for requests. You must specify **2484** for secure SSL connections, or **1521** for non-SSL connections

Note: The TCPS protocol entry in the **listener.ora** file must appear *after* the TCP protocol entry.

Example:

The following example shows ADDRESS_LIST configuration for an SMS node called “hostSMP”:

```
LISTENER=
  (DESCRIPTION_LIST =
  (DESCRIPTION=(ADDRESS_LIST=
     (ADDRESS=
         (PROTOCOL=IPC)
         (KEY=SMF)
         )))
  (DESCRIPTION=(ADDRESS_LIST=
     (ADDRESS=
         (PROTOCOL=TCP)
         (HOST=hostSMP)
         (PORT=1521)
         )))
  (DESCRIPTION=(ADDRESS_LIST=
     (ADDRESS=
         (PROTOCOL=TCPS)
         (HOST=hostSMP)
         (PORT=2484)
         )))
     )
)
```

Note: The ORACLE_SID for the SMF database is SMF. The listener can be made aware of this by adding an ADDRESS entry to the ADDRESS_LIST.

Step	Action

5	The listener also needs to know where it can find the information for any particular ORACLE_SID. This is accomplished through SID_LIST. The listener needs to know the name of the SID, the Oracle home directory and the global database name.
	Add an entry to SID_LIST by using the following syntax:
	`SID_LIST_LISTENER=(SID_LIST=`
	`    (SID_DESC=`
	`        (SID_NAME=SMF)`
	`        (ORACLE_HOME=oracle_home_directory)`
	`        (GLOBAL_DBNAME=SMF.Hostname)`
	`   )`
	`)`
	Where:
	• *oracle_home_directory* is the directory in which Oracle Database is installed
	• SMF.*Hostname* is the global database name. *Hostname* is the hostname of the SMS node
	Example
	The following example shows SID_LIST configuration for an SMS node called "hostSMP":
	`SID_LIST_LISTENER=(SID_LIST=`
	`    (SID_DESC=`
	`        (SID_NAME=SMF)`
	`        (ORACLE_HOME=/u01/app/oracle/product/12.1.0)`
	`        (GLOBAL_DBNAME=SMF.hostSMP)`
	`   )`
	`)`

6	Comment out the following entries:
	`USE_PLUG_AND_PLAY_LISTENER = TRUE`
	`USE_CKPFILE_LISTENER = TRUE`
	Important: Do not change the following settings:
	• `STARTUP_WAIT_TIME_LISTENER = 0`
	• `CONNECT_TIMEOUT_LISTENER = 10`

7	If you are using SSL connections to the database, set the following lines to these values:
	`SSL_CLIENT_AUTHENTICATION=FALSE`
	`SSL_CIPHER_SUITES=(TLS_RSA_WITH_AES_128_CBC_SHA)`
	Notes: You must also:
	• Configure the same entries for SSL_CLIENT_AUTHENTICATION and SSL_CIPHER_SUITES in the **sqlnet.ora** file.
	• Set the sslCipherSuites Java applet parameter in **sms.jnlp** and the SSL_CIPHER_SUITES entry to the same value.

8	Save and close the file.

9	Stop the listener and then restart the listener using the updated configuration by entering the following commands:
	`lsnrctl stop`
	`lsnrctl start`

Configuring Oracle Listener Java Applet Parameters

You configure the Java applet parameters for the Oracle listener in the **sms.jnlp** file. The installation process attempts to automatically configure this file for you, but you must check the data in the **sms.jnlp** file to ensure it is completely accurate.

Follow these steps to configure the Java applet parameters for the Oracle listener.

Step **Action**

1 Log on as user *root*.

2 Edit the **/IN/html/sms.jnlp** file by using a text editor such as vi; for example:

```
vi /IN/html/sms.jnlp
```

3 If you are using secure SSL connections to the database on a non-clustered system, configure the secureConnectionDatabaseHost applet parameter entry. The parameter value must be all on one line in the **.jnlp** file:

```
<param name="secureConnectionDatabaseHost" value="(DESCRIPTION=
(ADDRESS_LIST= (ADDRESS=(PROTOCOL=TCPS) (HOST=host_ip_addr)(PORT=lport)))
(CONNECT_DATA= (SERVICE_NAME=db_sid )))" />
```

If you are using secure SSL connections to the database on a clustered system, configure the `secureConnectionClusterDatabaseHost` applet parameter entry. The parameter value must be all on one line in the **.jnlp** file:

```
<param name="secureConnectionClusterDatabaseHost" value="(DESCRIPTION=
(ADDRESS_LIST= (ADDRESS=(PROTOCOL=TCPS) (HOST=host_ip_addr)(PORT=lport)))
(CONNECT_DATA= (SERVICE_NAME=db_sid )))" />
```

Where:

- *`host_ip_addr`* is the host name or IP address of the SMS node
- *`lport`* is the listener port for SSL connections using the TCPS protocol. Set LPORT to 2484 for SSL connections.
- *`db_sid`* is the database SID

In addition, for SSL connections the `EncryptedSSLConnection` Java applet parameter must be left undefined or set to true.

Example Java applet parameter configuration for SSL connections to the database (non-clustered)

```
<param name="secureConnectionDatabaseHost" value="(DESCRIPTION=
(ADDRESS_LIST= (ADDRESS=(PROTOCOL=TCPS) (HOST=hostSMP)(PORT=2484)))
(CONNECT_DATA= (SERVICE_NAME=SMF)))" />
<param name="EncryptedSSLConnection" value="true" />
```

4 If you are using SSL connections to the database you must set the `sslCipherSuites` Java applet parameter to TLS_RSA_WITH_AES_128_CBC_SHA:

```
<param name = "sslCipherSuites" value="(TLS_RSA_WITH_AES_128_CBC_SHA)" />
```

Step	Action

5	If you are using non-SSL connections to the database you must set the `EncryptedSSLConnection` parameter to false, and edit the following applet parameter entries:
	`<param name="host" value="host_ip_addr" />`
	`<param name="databaseID" value="lport:db_sid />`
	Where:
	• *`host_ip_addr`* is the host name or IP address of the SMS node
	• *`lport`*:*`db_sid`* is the listener port and the database SID. Set LPORT to 1521 for non-SSL connections.
	Example Java applet configuration for non-SSL connections to the database
	`<param name="HOST" value="hostSMP" />`
	`<param name="DATABASEID" value="1521:SMF" />`
	`<param name="EncryptedSSLConnection" value="false" />`

6	Save and close the file. The parameters in the **sms.jnlp** file are updated to reflect those of the Oracle listener.

Note: The **sms.html** file has been deprecated. However, if you upgraded from an earlier version of Convergent Charging Controller, you may continue to use the **sms.html** file. You must ensure that you set parameters to the same value in both the **sms.html** file, and the **sms.jnlp** file.

Configuring the SNMP Agent

Introduction

SNMP trap relaying is not automatically enabled. If you require SNMP trap relaying then you must perform the steps described in this topic.

The SNMP agent supports the following functionality:

- Forwarding of alarms as SNMP traps, using the Alarm Relay mechanism (see *Service Management System User's Guide*)
- Resynchronization of traps, enabling an SNMP manager to request resend of traps

Traps may be forwarded to multiple SNMP managers.

Note: This is subject to the following restrictions:

- All managers must use the same port to receive SNMP traps;
- All managers must be configured to use the same Community string
- Any triggering of the resynchronization mechanism results in duplicate traps being forwarded to all managers.

Configuring the snmp.cfg file

The SNMP agent is configured via the Alarm Notification screen and the snmp configuration file as described in this section. The configuration file is **/IN/service_packages/SMS/etc/snmp.cfg**.

The name of the network management station is defined by the destination field in the rule, used to match alarms. This allows alarms to be sent to multiple machines and also to determine which alarms should be sent to which machines. The SNMP-specific parameters are:

- `TARGET = “SNMP”`

- `DESTINATION = `*`manager_hostname`*

The other parameters are the same for all destinations and are determined from this configuration file, read at the start up of the smsAlarmRelay program.

In understanding these parameters, you must be familiar with the Simple Network Management Protocol (SNMP).

We currently support SNMP v3 (IETF STD0062). SNMP v1 (IETF RFC1157) traps are supported for backward compatibility purposes only.

To support integration with as broad a range of SNMP managers as possible, two forms of SNMP trap are supported:

- Opaque traps include all of the fault data in a single structured data type
- Multiple variable traps, wherein each fault datum is represented by a distinct trap variable

A single trap type must be chosen for each installation. See the "opaque" and "specific" configuration parameter descriptions below for details.

SNMP relaying - switching on

Follow these steps to turn on SNMP relaying of alarms.

Note: Like any command line switches, the `-p` can appear at any point in the command line. `-p` is a parameter without any options, and is used to enable SNMP relaying of alarms. SNMP relaying of alarms is off by default.

Step	Action
1	Open the **snmp.cfg** script with a text editor such as vi. The **snmp.cfg** file is located here by default: **/IN/service_packages/SMS/etc/snmp.cfg**
2	Add `-p` to the command line.
3	Save and close the file.

snmp.cfg example

This text shows the content of an example **snmp.cfg** file.

```
use-SNMPv3: 1
listenPort: 1161
userName: smf_oper
community: public
my-addr: addr
trap: 6
specific: 1
opaque: 1
port: 162
```

snmp.cfg file parameters

The parameters available in this file are described below. The only parameter that you are required to modify is “my-addr”; the rest are given for reference only.

Note: Separate the parameter from the value using the colon ‘:’.

`community`

Syntax:	`community: `*`type`*
Description:	The community to which smsAlarmRelay belongs.

Type:	String
Optionality:	Required
Allowed:	
Default:	public
Notes:	
Example:	`community: public`

`listenPort`

Syntax:	`listenPort:` *`port`*
Description:	The UDP port number from which smsAlarmRelay listens for `get-` and `set-variable` requests.
Type:	Integer
Optionality:	Required
Allowed:	1 - 65535
Default:	
Notes:	If the `use-SNMPv3` parameter is set to 0, the `listenPort` parameter has no effect.
Example:	`listenPort: 1161`

`my-addr`

Syntax:	`my-addr:` *`addr`*
Description:	The Internet Protocol (IP) address of the computer on which smsAlarmRelay is installed.
Type:	String
Optionality:	Required
Allowed:	May be either a symbolic host name or an Internet protocol number expressed in dotted-decimal format.
Default:	
Notes:	In SNMP terminology, *addr* is called agent-addr. Most hosts have at least two addresses, the second one being the loop-back address: 127.0.0.1.
Example:	A symbolic host name might be `SMS_main_1`. `my-addr: SMS_main_1` An Internet protocol number could be `192.0.2.0`. `my-addr: 192.0.2.0`

`my-oid`

Syntax:	`my-oid:` *`id`*
Description:	The alarm parameter argument assigned to Oracle.
Type:	String
Optionality:	Optional (deprecated)
Allowed:	1.2.36.52947743
Default:	1.2.36.52947743
Notes:	
Example:	

`notification-oid`

Syntax:	`notification-oid: `*`str`*
Description:	A variable that can be queried or changed remotely.
Type:	String
Optionality:	Optional (deprecated)
Allowed:	Constructed from the value of the `param-oid` parameter to which is appended two additional digits. The value of each digit is determined by the format of alarms.

1.2.36.52947743.1.1	Opaque encoding of Oracle fields.
1.2.36.52947743.1.2	id, .3 = machine, .4 = time, .5 = cpu, etc.
1.2.36.52947743.2.1	Opaque encoding of X.733 fields.
1.2.36.52947743.2.2	Managed object instance, .3 event type, etc.

Default: 1.2.36.52947743.2.1

Notes: The notification-oid parameter requires that:

- The `use-SNMPv3` parameter is set for SNMP version 3.
- The `listenPort` parameter is configured.

Example:

`opaque`

Syntax:	`opaque: 0|1`
Description:	Defines encoding for SNMP specific traps.
Type:	Boolean
Optionality:	Required if the `specific` (on page 74) parameter is used.
Allowed:	0 Use if `specific` is set to 2 or 4.
	1 Use if `specific` is set to 1 or 3.
Default:	
Notes:	The value depends on the value assigned to the `specific` parameter.
Example:	`opaque: 1`

`param-oid`

Syntax:	`param-oid: `*`id`*
Description:	The alarm parameter argument assigned to Oracle.
Type:	String
Optionality:	Optional (deprecated)
Allowed:	1.2.36.52947743
Default:	1.2.36.52947743
Notes:	The *id* is constructed from the values of the sub-parameters listed below.

iso	country	australia	Oracle
1	2	36	52947743

Example:

`port`

Syntax:	`port:` *`port`*
Description:	The Internet Protocol (IP) port number of the remote SNMP manager computer.
Type:	Integer
Optionality:	Required
Allowed:	1 - 65535
Default:	162
Notes:	162 is the SNMP trap port.
Example:	`port: 162`

`specific`

Syntax:	`specific:` *`int`*	
Description:	An SNMP-specific trap parameter.	
Type:	Integer	
Optionality:	Required if you set the `opaque` parameter	
Allowed:	1	A single opaque binding.
	2	Multiple variable bindings per trap, for each parameter.
	3	A single opaque binding in x733 format.
	4	Multiple x733 variable bindings per trap.
Default:		
Notes:	If you use the `specific` parameter, you must also set the `opaque` parameter.	
Example:	`specific: 1`	

`trap`

Syntax:	`trap: int`
Description:	The value of the generic trap.
Type:	Integer
Optionality:	Required
Allowed:	6
Default:	6
Notes:	
Example:	`trap: 6`

`use-SNMPv3`

Syntax:	`use-SNMPv3:` *`0|1`*	
Description:	The version of the SNMP implementation.	
Type:	Integer	
Optionality:	Required	
Allowed:	0	SNMPv1 is enabled
	1	SNMPv3 is enabled
Default:	1	
Notes:		
Example:	`use-SNMPv3: 1`	

`userName`

Syntax:	`userName:` *`name`*
Description:	Used by smsAlarmRelay when it listens on a standard SNMP port that has already been opened.
Type:	String
Optionality:	Required
Allowed:	
Default:	smf_oper
Notes:	In order to open the standard SNMP port, smsAlarmRelay needs root privileges. Once the port is open, smsAlarmRelay's privileges are restricted to those assigned to *`name`*.
Example:	`userName: smf_oper`

Formatting an SNMP trap message

The format of SNMP messages is defined in IETF STD0062.

At the top level the “Message” element has the “version” field set in accordance with the SNMP version set by the "use-SNMPv3" configuration parameter. The rest of the formatting differs according to the SNMP version that is being used.

SNMP v1

The SNMP v1 message is built up from each line of this table.

Part	Set from
version	Set by the use-SNMPv3 configuration parameter.
community	Set via the community configuration parameter.
enterprise	Set using the my-oid configuration parameter.
agent-addr	The IP address of the SMS set using my-addr parameter.
generic-trap	Set using the trap configuration parameter.
specific-trap	Set using the the specific parameter.

SNMP v3

The SNMP v3 message is built up as follows.

- version - set by the use-SNMPv3 configuration parameter
- Global Header - including a usm security model
- security parameters
  - authoritative Engine ID - security ID
  - engine boots - record of the number of boots of the alarmRelay
  - engine time - record of the up of the alarmRelay
- context engine ID - PID of smsAlarmRelay
- context name - "smsAlarmRelay"
- v2 trap PDU
  - error status
  - error index

variable bindings

The variable-bindings take one of two forms, in accordance with the settings of the `opaque` (on page 73) and `specific` (on page 74) configuration parameters.

The opaque form is composed of a sequence containing a single item. That single item is itself a sequence comprising of a pair. The pair is the object ID of the alarm (obtained from the configuration file) and the alarm data itself encased as an “Opaque” data item.

The multiple variable form is composed of a sequence of pairs, each pair being an object ID identifying the variable and the variable values. The object IDs and variable datatypes are specified in the MIB.

See *SMF AlarmMessage Format* (see "*Configuring the SNMP Agent*" on page 70, on page 79) for the ASN.1 format of the alarm data.

Transmission of the SNMP trap message

Given the trap message that has been previously formatted we can now send it to the network management station. As defined in RFC 1157, the message is sent over the User Datagram Protocol (UDP). The destination IP address and the port are specified in the configuration file.

Failure to send the trap does not raise an alarm as this would lead to an infinite loop of alarm messages.

Starting and stopping

The SNMP additions to the smsAlarmRelay send a “start” trap to all configured destinations when it starts up. Similarly, it sends a “stopped” trap and process shutdown.

Restarting the smsAlarmRelay

By default, SNMP trap relaying is not performed. Therefore the **smsAlarmRelayStartup.sh** script must be edited and the `smsAlarmRelay` (on page 99) process restarted using the steps below.

Follow these steps to restart the `smsAlarmRelay` daemon.

Step	Action
1	Type following command to find the process ID: `ps -ef | grep smsAlarmRelay` **Note:** The second column of the results returned is the process ID and the third column gives the parent process ID. Kill the process ID from the second column.
2	Type `kill -TERM` *`pid`* **Result:** The process is terminated and is restarted by the inittab process.

Configuring Connections for CORBA Services

About CORBA Services Configuration

The CorbaServices section in the **eserv.config** configuration file defines common connection parameters for CORBA services on SMS nodes. The CorbaServices configuration overrides the default and command-line values specified for CORBA listen ports and addresses.

If you are using IP version 6 addresses, then you must include the CorbaServices section in the **eserv.config** file on SMS nodes. This section is optional if you are using only IP version 4 addresses.

The CorbaServices section includes the following required parameters:

- AddressInIOR
- smsTaskAgentOrbListenPort
- smsReportDaemonOrbListenPort
- smsTrigDaemonOrbListenPort
- ccsBeOrbListenPort

Example CORBA Services Configuration on the SMS

The following example shows the CorbaServices configuration section in the **eserv.config** file for CORBA services on the SMS node.

```
CorbaServices = {
    AddressInIOR = "sms_machine.oracle.com"
    OrbListenAddresses = [
        "2001:db8:0:1050:0005:ffff:ffff:326b"
        "192.0.2.0"
    smsTaskAgentOrbListenPort = 6332
    smsReportDaemonListenPort = 6333
    smsTrigDaemonOrbListenPort = 6334
    ccsBeOrbListenPort = 6335
}
```

CorbaServices Parameters

You specify CORBA services configuration in the CorbaServices section of the **eserv.config** file on SMS and SLC nodes. The CorbaServices configuration supports the following parameters:

AddressInIOR

Syntax:	`AddressInIOR = "`*`str`*`"`
Description:	The hostname or IP address to place in the IOR (Interoperable Object Reference) for the CORBA service.
Type:	String
Optionality:	Required (on SMS nodes only)
Allowed:	Hostname, IP version 6 address, or IP version 4 address
Default:	
Notes:	

Examples:	`AddressInIOR = "2001:db8:0:1050:0005:ffff:ffff:326b"`
	`AddressInIOR = "192.0.2.0"`
	`AddressInIOR = "sms03xxx.us.oracle.com"`

OrbListenAddresses

Syntax:	`OrbListenAddresses = [`
	`"`*`str`*`"`
	`["str"]`
	`]`

Description:	List of IP addresses on which the CORBA service listens for incoming requests.
Type:	Array
Optionality:	Optional (on SMS nodes only)
Allowed:	IP version 6 addresses, and IP version 4 addresses

Default:	
Notes:	If the `OrbListAddresses` parameter is not set, or you do not specify any IP addresses, then the CORBA service listens on all the IP addresses available on the host. Loopback IP addresses and special IP addresses, as defined in RFC 5156, are excluded.
Example:	`OrbListenAddresses = [ "2001:db8:0:1050:0005:ffff:ffff:326b" "192.0.2.0" ]`

```
OrbListenAddresses = [
    "2001:db8:0:1050:0005:ffff:ffff:326b"
    "192.0.2.0"
]
```

smsTaskAgentOrbListenPort

Syntax:	`smsTaskAgentOrbListenPort =` *`int`*
Description:	The number of the port on which smsTaskAgentOrb listens.
Type:	Integer
Optionality:	Required (on SMS nodes only)
Allowed:	
Default:	
Notes:	Overrides the CORBA service port specified for the smsTaskAgent process in the `-s` command-line parameter. For more information, see *smsTaskAgent* (on page 139).
Example:	`smsTaskAgentOrbListenPort = 6332`

smsReportDaemonOrbListenPort

Syntax:	`smsReportDaemonOrbListenPort =` *`int`*
Description:	The number of the port on which smsReportDaemonOrb listens.
Type:	Integer
Optionality:	Required (on SMS nodes only)
Allowed:	
Default:	
Notes:	Overrides the CORBA listen port specified for the smsReportDaemon process in the `-s` command-line parameter. For more information about smsReportDaemon, see *smsReportsDaemon* (on page 122).
Example:	`smsReportDaemonOrbListenPort = 6333`

smsTrigDaemonOrbListenPort

Syntax:	`smsTrigDaemonOrbListenPort =` *`int`*
Description:	The number of the port on which smsTrigDaemonOrb listens.
Type:	Integer
Optionality:	Required (on SMS nodes only)
Allowed:	
Default:	
Notes:	Overrides the smsTrigDaemon CORBA listen port set in the `listenPort` parameter in the triggering section of the **eserv.config** file. For more information about smsTrigDaemon, see *smsTrigDaemon* (on page 143).
Example:	`smsTrigDaemonOrbListenPort = 6334`

ccsBeOrbListenPort

Syntax:	`ccsBeOrbListenPort =` *`int`*
Description:	The number of the port on which ccsBeOrb listens.

Type:	Integer
Optionality:	Required (on SMS nodes only)
Allowed:	
Default:	
Notes:	Overrides the CORBA listen port specified for the ccsBeOrb process in the `listenPort` parameter. For more information, see *Charging Control Services Technical Guide*.
Example:	`ccsBeOrbListenPort = 6335`

SMF AlarmMessage Format

Introduction

This topic provides the format of the SMFalarmMessage including the MIB definitions.

Alarm Table fields

This table defines the layout of the SMF_ALARM_MESSAGE and SMF_ALARM_DEFN tables in the SMF from which the alarms are derived.

Name	Field size	Field type	Null value
id	38	NUMBER	not null
machine	16 (15 characters for hostname; 1 terminating charater)	VARCHAR2	not null
time		DATE	not null
cpu	3	NUMBER	not null
name	6	NUMBER	not null
subsystem	24	VARCHAR2	not null
severity	1	NUMBER	not null
description	256	VARCHAR2	
opcomment	256	VARCHAR2	
count	4	NUMBER	not null
close_time		DATE	
status	7	VARCHAR2	
change_sequence	38	NUMBER	
managed_object_instance	2000	VARCHAR2	
event_type	2	NUMBER	
probable_cause	4	NUMBER	
specific_problem	256	VARCHAR2	
perceived_severity	1	NUMBER	
additional_text	1000	VARCHAR2	

MIB field mappings - SMF_ALARM_MESSAGE

This table provides the SMF_ALARM_MESSAGE to MIB field mappings.

DB Alarm		MIB
0	id	Mapped directly (unique ID)
1	machine	Mapped directly (hostname)
2	time	Mapped directly (“YYYYMMDDHHMMSS”
3	cpu	Mapped directly (CPU number)
4	name	= 0 for Solaris & HPUX
6	subsystem	Mapped directly (process identifier)
7	severity	Mapped directly (0=NOTICE, 2=WARNING, 4=ERROR, 6=CRITICAL, 8=CLEARANCE)
8	description	Mapped directly (free text)
9	opcomment	Mapped directly (free text)
10	count	Mapped directly (number of duplicates)
	close_time	Not sent
5	status	Mapped directly (“OPEN”, “PENDING”, “CLOSED”)
	change_sequence	Not sent

MIB field mappings - SMF_ALARM_MESSAGE

This table provides the SMF_ALARM_MESSAGE to MIB field mappings.

DB Alarm		MIB
0	id	Mapped directly (unique ID)
1	machine	Mapped directly (hostname)
2	time	Mapped directly (“YYYYMMDDHHMMSS”
3	cpu	Mapped directly (CPU number)
4	name	= 0 for Solaris
6	subsystem	Mapped directly (process identifier)
7	severity	Mapped directly (0=NOTICE, 2=WARNING, 4=ERROR, 6=CRITICAL, 8=CLEARANCE)
8	description	Mapped directly (free text)
9	opcomment	Mapped directly (free text)
10	count	Mapped directly (number of duplicates)
	close_time	Not sent
5	status	Mapped directly (“OPEN”, “PENDING”, “CLOSED”)
	change_sequence	Not sent

MIB field mappings - SMF_ALARM_DEFN

This table provides the SMF_ALARM_DEFN to MIB field mappings.

DB Alarm	MIB
Alarm_type_id	not sent

DB Alarm	MIB
event_type	Mapped directly (event_type)
probable_cause	Mapped directly (probable_cause)
severity	Mapped directly (severity)
specific_problem	Mapped directly (specific_problem)
recommended_action	not sent
additional_text	Prefixed with description and mapped to additional_text
present_to_am	not sent
present_to_ar	not sent
autoclear period	not sent
regular_expression	not sent
notes	not sent

SMF Listen Messages

An SNMP manager may trigger the resend of traps by setting eServDataLastChgSeq to the value of the identifier (id or eServId) of the last successfully received trap.

Note: Use of this mechanism will cause traps to be sent to all active SNMP managers.

Defining the Screen Language

Introduction

The default language file sets the language that the Java administration screens start in. The user can change to another language after logging in.

The default language can be changed by the system administrator.

By default, the language is set to English. If English is your preferred language, you can skip this step and proceed to the next configuration task, *Defining the Help Screen Language* (on page 83).

Default.lang

When SMS is installed, a file called **Default.lang** is created in the application's language directory in the screens module. This contains a soft-link to the language file that defines the language used by the screens.

If a **Default.lang** file is not present, the **English.lang** file is used.

The SMS **Default.lang** file is:

/IN/html/SMS/language/Default.lang

Example Screen Language

If Dutch is the language you want to set as the default, create a soft-link from the **Default.lang** file to the **Dutch.lang** file.

Language files for multi-byte character sets

To create and use a language file for a language that requires a multi-byte character set, as simplified or traditional Chinese does, as well as others, you should create the file in the UTF-8 (Unicode Transformation Format-8) format.

Note:To support reading and writing of UTF-8 characters, you must ensure that the database character set is UTF-8. You can use the following query to determine what the database character set is:

```
select value from nls_database_parameters where parameter =
'NLS_CHARACTERSET';
```

User-specific language settings

All screens in the SMS are able to support selected languages. On login, the screens are displayed in the default language. You can subsequently specify a language for a specific user in the **Configuration** field of the **User Management** screen by specifying LANGUAGE=*ABC* where *ABC* must match the language file name, is case-sensitive, and does not include the file name extension. After a language is selected for a user, it is stored in their profile.

If a character set other than UTF-8 is used to create the language file, you must specify the character set for a user using CHARSET=*XYZ* in the Configuration field on the **User** tab of the **User Management** screen, where *XYZ* specifies one of the following character sets: US-ASCII, ISO-8859-1, UTF-16BE, UTF-16LE, or UTF-16.

For more information about setting the Configuration field, see *Service Management System User's Guide*.

Procedure

Follow these steps to set the default language for your SMS Java Administration screens.

Step	Action
1	Change to the following directory: `/IN/html/SMS/language` **Example command:** `cd /IN/html/SMS/language/`
2	Ensure the Default.lang file exists in this directory.
3	If the required file does not exist, create an empty file called **Default.lang.**

4	Ensure that the language file for your language exists in this directory. The file should be in the format:
	`language.lang`
	Where:
	language = your language.
	Example:
	`Spanish.lang`

5	If the required language file does not exist, either: • create a new one with your language preferences, or • contact Oracle support. To create a language file, you need a list of the phrases and words used in the screens. These should appear in a list with the translated phrase in the following format: `original phrase=translated phrase` Any existing language file should have the full set of phrases. If you do not have an existing file to work from, contact Oracle support with details.
6	Create a soft-link between the **Default.lang** file, and the language file you want to use as the default language for the SMS Java Administration screens.

Step	Action
	Example command: `ln -s Dutch.lang Default.lang`

Defining the Help Screen Language

Introduction

The default Helpset file sets the language that the help system for the Java Administration screens start in. The user can change to another language after logging in.

The default language can be changed by the system administrator. By default, the language is set to English.

Default.SMS.hs

When SMS is installed, a file called **Default.SMS.hs** is created in the application's language directory in the screens module. This contains a soft-link to the language file which defines the language which will be used by the screens.

If a **Default.SMS.hs** file is not present, the English.SMS.hs file will be used.

If a **Default.SMS.hs** file is present, a user must explicitly set their language to their required language in the Tools screen or the default language will be used.

The **Default.SMS.hs** file is:

```
/IN/html/SMS/helptext/Default.SMS.hs
```

Example helpset language

If Dutch is the language you want to set as the default, create a soft-link from the **Default.SMS.hs** file to the **Dutch.SMS.hs** file.

Procedure

Follow these steps to set the default language for your SMS Java Administration screens.

Step	Action
1	Change to the following directory: `/IN/html/SMS/helptext` **Example command:** `cd /IN/html/SMS/helptext`
2	Ensure the **Default.SMS.hs** file exists in this directory.
3	If the required file does not exist, create an empty file called **Default.SMS.hs**.
4	Ensure that the language file for your language exists in this directory. The file should be in the format: *`language`*`.SMS.hs` **Where:** *language* is your language **Example:** `Dutch.SMS.hs`
5	If the required language file does not exist, either: • create a new one with your language preferences, or • contact Oracle support.

To create a language file, you need a list of the phrases and words used in the screens. These should appear in a list with the translated phrase in the following format:

```
original phrase=translated phrase
```

Any existing language file should have the full set of phrases. If you do not have an existing file to work from, contact Oracle support with details.

6 Create a soft-link between the **Default.SMS.hs** file, and the language file you want to use as the default language for the SMS Java Administration screens.

Example command: `ln -s Dutch.Acs_Service.hs Default.Acs_Service.hs`

Setting up the Screens

Accessing SMS

On first use of the application, the SMS user interface (UI) needs to be set up on your local machine. Use one of the following methods to open the UI from the relevant web page:

- Java Webstart, open the Service Management System default page on the *SMS_hostname*, then click the **WebStart** link.
- SMS Webstart directly:

```
http://SMS_hostname/sms.jnlp
```

where:

SMS_hostname is the hostname of the SMS node.

Result: The SMS UI starts.

For more information about the SMS UI, see *Service Management System User's Guide*.

About Assigning the Oracle Profile to New Users

You create the users who can access the SMS UI through the Service Management System, User Management screen. By default, when you add a new SMS user, the new user is assigned the standard Oracle profile, named DEFAULT. This profile includes a password verification function that is applied to user passwords and that determines the specific conditions for a valid password, such as the minimum length, number of digits, and so on.

You can create a non-standard Oracle profile to assign to new users by using the CREATE PROFILE command. When you create the Oracle profile, you specify the password verification function that will be applied to user passwords. You can use this feature, for example, to specify an Oracle profile that uses a password verification function that has stricter password verification conditions.

For information about creating Oracle profiles by using the CREATE PROFILE command, see the Oracle Database online documentation.

When you create or edit a user's password, smsTaskAgent verifies that you have entered an acceptable password by applying the password verification function that is specified in the Oracle profile assigned to the user.

You configure smsTaskAgent to assign a non-standard Oracle profile to new users instead of the default Oracle profile as follows:

```
smsTaskAgent = {
    defaultOracleProfile = "password_profile"
}
```

where *password_profile* is the name of the Oracle profile you want to use. You must specify the name of an existing Oracle profile. See *smsTaskAgent* (on page 139) for more information about smsTaskAgent.

You specify the message that displays for failed attempts to create or change a user's password in the `passwordPolicyMessage` Java applet parameter. See *passwordPolicyMessage* (on page 89) for more information.

About customizing the SMS UI

You can customize the SMS UI by setting applet parameters in the **sms.jnlp** file located in the **/IN/html/** directory. You set jnlp applet parameters by using the following syntax:

```
<param name="parameter" value="value" />
```

Where:

- *`parameter`* is the name of the applet parameter
- *`value`* is the value to which that parameter is set

About Applet Parameters in .html Files

The ability to customize the Convergent Charging Controller UI by setting applet parameters in the following **.html** files has been deprecated:

- **acs.html**
- **sms.html**
- **vpn.html**

If you upgraded from an earlier version of Convergent Charging Controller, you may continue to set applet parameters in these files. However, you must ensure that any parameters that you set are also set to the same value in the corresponding **.jnlp** file:

- **acs.jnlp**
- **sms.jnlp**
- **vpn.jnlp**

Note: You use the following syntax to set applet parameters in the **.html** files:

```
<param name=parameter value="value">
```

Where:

- *`parameter`* is the name of the Java applet parameter
- *`value`* is the value to which that parameter is set

Resource properties

The following resource properties in the resources section of the **sms.jnlp** file define the Java 2 SE runtime environment:

`java_arguments`

Syntax:	`<property name="jnlp.packEnabled" value="true|false" />`
Description:	This Java resource parameter enables the use of the pack jar sig file when set to true.
Type:	Boolean
Optionality:	Required
Allowed:	• true – The Java plug-in automatically downloads and uses the compressed jar file (**sms.sig.jar.pack.gz**). If the browser is unable to use the compressed jar file, the uncompressed jar file (**sms.jar.sig**) is downloaded. • false – Downloads the **sms.jar.sig** file.

Default:	true
Notes:	Use of the compressed file improves the speed of launching the application. For more information, refer to Oracle Java SE Documentation
Example:	`<property name="jnlp.packEnabled" value="true" />`

`j2se version`

Syntax:	`<j2se version="`*`version`*`" href="`*`J2_url`*`" />`
Description:	This Java resource parameter specifies the minimum Java 2 SE Runtime Environment (JRE) version that the application is supported on, and the URL for Java 2 SE.
Type:	String
Optionality:	Required
Allowed:	
Default:	
Notes:	For more information, see Oracle Java SE Documentation.
Example:	`<j2se version="1.6.0+" href="http://java.sun.com/products/autodl/j2se" />`

Java Applet Parameters

The following applet parameters are available to customize the UI:

`clusterDatabaseHost`

Syntax:	See example
Description:	This specifies the connection string (including a host and an alternative host address, in case the first IP address is unavailable) for non-SSL cluster-aware connection to the database. The standard TCP port **1521** in the ADDRESS_LIST line is used by default. To use non-SSL connections to the database, ensure that the `EncryptedSSLConnection` parameter is set to false.
Type:	String
Optionality:	Optional
Allowed:	
Default:	
Notes:	If present, this parameter will be used instead of the `databaseID` parameter.

Example:	`<param name="clusterDatabaseHost" value =`
	`"(DESCRIPTION=(LOAD_BALANCE=YES)(FAILOVER=ON)(ENABLE=BROKEN)`
	`(ADDRESS_LIST=(ADDRESS=(PROTOCOL=TCP)(HOST=`*`smsphysnode1`*`)(PORT=1521))`
	`(ADDRESS=(PROTOCOL=TCP)(HOST=`*`smsphysnode2`*`)(PORT=1521)))`
	`(CONNECT_DATA=(SERVICE_NAME=SMF)(FAILOVER_MODE=(TYPE=SESSION)(METHOD=BASIC)(RETRIES=5)(DELAY=3))))" />`

`databaseHost`

Syntax:	`<name="databaseHost" value = "`*`ip`*`:`*`port`*`:`*`sid`*`" />`
Description:	Sets the IP address and port to use for non-SSL connections to the SMF database, and the database SID. To use non-SSL connections to the database, you must specify to use the standard TCP port **1524,** and ensure that the `EncryptedSSLConnection` parameter is set to false. If the `EncryptedSSLConnection` parameter is undefined or set to true then the `databaseHost` parameter is not used.
Type:	String
Optionality:	Optional
Allowed:	
Default:	Not set. Secure SSL connection is enabled at installation by default.
Notes:	Internet Protocol version 6 (IPv6) addresses must be enclosed in square brackets []; for example: **`[2001:db8:`***`n`*`:`*`n`*`:`*`n`*`:`*`n`*`:`*`n`*`:`*`n`***`]`** where *`n`* is a group of 4 hexadecimal digits. The industry standard for omitting zeros is also allowed when specifying IP addresses.
Examples:	`<param name="databaseHost" value = "`**`192.0.2.1:2484:SMF`**`" />` `<param name="databaseHost" value = "`**`[2001:db8:0000:1050:0005:0600:300c:326b]:2484:SMF`**`" />` `<param name="databaseHost" value = "`**`[2001:db8:0:0:0:500:300a:326f]:2484:SMF`**`" />` `<param name="databaseHost" value = "`**`[2001:db8::c3]:2484:SMF`**`" />`

`databaseID`

Syntax:	`<name="databaseID" value="`*`port`*`:`*`sid`*`" />`
Description:	The number of the SQLNET port to use to connect to the database, and the database SID.
Type:	String
Optionality:	Required
Allowed:	
Default:	1521:SMF
Notes:	Set the `databaseID` port number to **1521** and set the `Encrypted SSLConnection` parameter to false to use non-SSL connections to the database. If the `EncryptedSSLConnection` is undefined or set to true, then `databaseID` is not used. Instead, one of `secureConnectionDatabaseHost` or `secureConnectionClusterDatatbaseHost` is used for secure SSL connection to the database.
Example:	`<param name="databaseID" value="1521:SMF" />`

`DUAL_STATS_NODE`

Syntax:	See example
Description:	Specifies whether the **View Statistics** tab of the Statistics Viewer screen displays the SMS node.
Type:	String
Optionality:	Optional

Allowed:	• true – The **View Statistics** tab of the Statistics Viewer screen displays information about the SMS node. • false – The **View Statistics** tab of the Statistics Viewer screen does not display information about the SMS node.
Default:	false
Notes:	For more information, see Viewing Statistics in *Service Management System User's Guide*.
Example:	`<param name="DUAL_STATS_NODE" value="true" />`

EncryptedSSLConnection

Syntax:	See example
Description:	Enables secure encrypted SSL connections to the database through the UI.
Type:	Boolean
Optionality:	Optional (default used if not set)
Allowed:	true – Use encrypted SSL connections to access the UI false – Use non-SSL connections to access the UI
Default:	true
Notes:	When set to true, the `databaseID` parameter is ignored and either the `secureConnectionDatabaseHost` parameter or the `secureConnectionClusterDatabaseHost` parameter must be set. When set to false, the listener port value in the `databaseID` parameter must be set to 1521.
Example:	`<param name="EncryptedSSLConnection" value = "true" />`

logo

Syntax:	See example
Description:	Sets the system graphic that displays briefly in a splash window immediately before the Login window displays. At installation, the logo parameter is set to the **.gif** file for the Oracle logo.
Type:	String
Optionality:	Optional
Allowed:	A valid network path/file.
Default:	None
Notes:	Set to the relative path name for the logo **`.gif`** file to use.
Example:	`<param name="logo" value="`**`SMS/images/oracle.gif`**`" />`

MAX_CONTROL_PLANS_DISPLAYED

Syntax:	See example
Description:	Sets the maximum number of control plans visible in the search box.
Type:	String
Optionality:	Optional
Allowed:	
Default:	
Notes:	
Example:	`<param name="MAX_CONTROL_PLANS_DISPLAYED" value="200" />`

`passwordPolicyMessage`

Syntax:	`<param name=passwordPolicyMessage value="`*`message_text`*`">`
Description:	The message text that is displayed for failed attempts to change a user's password through the User Management screen in the Service Management System UI.
Type:	String
Optionality:	Optional (default used if not set)
Allowed:	Any text. The text should be relevant to the password restrictions imposed by the password verification function defined in the user's (Oracle) profile.
Default:	The new password is not compliant with the password policy.
Notes:	The definition must be specified on one line. Do not include new lines in the message text. If the message is longer than 80 characters, the displayed message is broken up into multiple lines automatically.
Example:	`<param name=passwordPolicyMessage value="The new password must contain at least 9 characters and must contain at least 2 digits">`

`secureConnectionClusterDatabaseHost`

Syntax:	See example
Description:	The connection string (including host address and port) for secure SSL connections to the SMF database on a clustered system. To enable secure SSL connections to the database, you must specify to use the standard TCPS port **2484**, and ensure that the `EncryptedSSLConnection` parameter is set to true.
Type:	String
Optionality:	Optional (default used if not set)
Allowed:	
Default:	
Notes:	If present, this parameter will be used instead of the `secureConnectionDatabaseHost` parameter.

Example:	`<param name="secureConnectionClusterDatabaseHost" value = "`
	`(DESCRIPTION=`
	`(ADDRESS_LIST=(ADDRESS=(PROTOCOL=TCPS)(HOST=192.0.1.1)(PORT=2484))`
	`(ADDRESS=(PROTOCOL=TCP)(HOST=192.0.2.1)(PORT=2484)))`
	`(CONNECT_DATA=(SERVICE_NAME=SMF)))" />`

`secureConnectionDatabaseHost`

Syntax:	See example
Description:	The connection string (including host address and port) for secure SSL connections to the SMF database on a non-clustered system. To enable secure SSL connections to the database, you must specify to use the standard TCPS port **2484**, and ensure that the `EncryptedSSLConnection` parameter is set to true.
Type:	String
Optionality:	Optional (default used if not set)
Allowed:	
Default:	

Notes:	If present, this parameter will be used instead of the `databaseID` parameter.
Example:	`<param name="secureConnectionDatabaseHost" value = "(DESCRIPTION=(ADDRESS_LIST=(ADDRESS=(PROTOCOL=TCPS)(HOST=192.0.1.1)(PORT=2484))))(CONNECT_DATA=(SERVICE_NAME=SMF)))" />`

`sslCipherSuites`

Syntax:	See example
Description:	Sets the ciphers suites to use for SSL encryption. You must set this parameter if you are using secure SSL encryption to access the SMS database.
Type:	String
Optionality:	Optional (default used if not set)
Allowed:	(TLS_RSA_WITH_AES_128_CBC_SHA)
Default:	(TLS_RSA_WITH_AES_128_CBC_SHA)
Notes:	You must also set the SSL_CIPHER_SUITES parameter to (TLS_RSA_WITH_AES_128_CBC_SHA) in the **listener.ora** and **sqlnet.ora** files.
Example:	`<param name = "sslCipherSuites" value="(TLS_RSA_WITH_AES_128_CBC_SHA)" />`

`TZ`

Syntax:	See example
Description:	The screens in the Convergent Charging Controller UI display all time and date values in the time zone set by this parameter.
Type:	String
Optionality:	Optional (default used if not set)
Allowed:	Any Java supported time zone.
Default:	GMT
Notes:	For a full list a Java supported time zones see Time Zones.
Example:	`<param name="TZ" value="GMT" />`

Java Webstart

To launch GUI applications using Java Webstart, you must ensure the Web server supports the jnlp file type.

For example, if the Web server is Apache, find the **mime.types** file (default location is **/etc/apache/mime.types**), add the following line in the file, then restart Apache.

```
application/x-java-jnlp-file jnlp
```

Example jnlp resources and applet parameters

Here is an example of the parameters for resources and applet in the **sms.jnlp** file. Note that other applications, such as ACS and CCS may add parameters to this file.

```
<jnlp spec="1.0+"
   codebase="http://HOST_IP_ADDR/"
   href="sms.jnlp" >
   .
   .
   .

   <resources>
      <j2se version="1.6.0+" href="http://java.sun.com/products/autodl/j2se" />
      <property name="jnlp.packEnabled" value="true" />
```

```
        <jar href="sms.sig.jar" />
        <jar href="common.sig.jar" />
        <jar href="ojdbc6.jar.sig" />
        <extension name="Oracle Help for Java" href="ohj.jnlp" />
        .
        .
      <property name="java.util.Arrays.useLegacyMergeSort" value="true" />
    </resources>

    <applet-desc
        documentBase="http://HOST_IP_ADDR"
        name="SMSApplet"
        main-class="UserScreens.Applet1"
        width="275"
        height="25" >
        <param name="TZ" value="GMT" />
        <param name="host" value="HOST_IP_ADDR" />
        <param name="logo" value="SMS/images/oracle.gif" />
        <param name="databaseID" value="LPORT:DB_SID" />
        <param name="secureConnectionDatabaseHost" value="(DESCRIPTION=(ADDRESS_LIST=
        (ADDRESS=(PROTOCOL=TCPS) (HOST=HOST_IP_ADDR) (PORT=2484)))
        (CONNECT_DATA= (SERVICE_NAME=SMF)))"/>
        <param_name="sslCipherSuites" value="(TLS_RSA_WITH_AES_128_CBC_SHA)" />
        <param_name="EncryptedSSLConnection" value="true" />
        <param name="INProtocol" value="" />
        <param name="UseAnnouncements" value="YES" />
        <param name="configPlugin" value="/Configuration/plugins/esgConfig/plugin.xml" />
        <param name="workingConfigFile" value="/Configuration/xml/esgConfigWorking.xml" />
        <param name="masterConfigFile" value="/Configuration/xml/esgConfigMaster.xml" />
    </applet-desc>


</jnlp>
```

Configuring Nodes

SMS Nodes

During installation of the SMS software, each SMS is set up so that it is a valid replication node. Check that each node has at least the following configuration details:

- Valid primary address (or hostname)
- Node number of 1-16 (starting at 1), and
- Validator check box checked.

You can check the setup via the Node Management screen in the SMS Administration screens. For more information on node configuration and setup, see *Service Management System User's Guide*.

SLC Nodes

In a clustered installation, each SLC has one node number associated with it:

- One in the range 256 to 511 for the Update Loader

These node numbers can be assigned using the **Node Management** screen in the SMS Java screens.

Each Update Loader should at least have:

- Valid primary address (or hostname)
- Node number in the range 256 to 511 (the Node Numbers of the Update Loader should start at 301).
- Empty validator check box.

For more information on node configuration and setup, see the *SMS User's Guide*.

Statistics nodes

You must complete the process by configuring Statistics within the SMS, see *SMS User's Guide*.

Installing Additional Applications

Installing the applications

Follow these steps to install the applications.

Step	Action
1	Install each application to create a set of replication groups.
2	Decide which SLCs will run this application.
3	Target all of the new groups (or such different set as is advised in the instructions for the application) onto each of these SLCs (the i+256 node).

Order of replication

Please note that the order in which replication tables are added is important.

Chapter 5

Background Processes on the SMS

Overview

Introduction

This chapter provides a description of the programs or executables used by the System as background processes on an SMS.

Executables are located in the `/IN/service_packages/SMS/bin` directory.

Some executables have accompanying scripts that run the executables after performing certain cleanup functions. All scripts should be located in the same directory as the executable.

For more information about the processes and systems that use these programs and executables, see *System Overview* (on page 1).

Important: It is a pre-requisite for managing these core service functions that the operator is familiar with the basics of Unix process scheduling and management. Specifically, the following Unix commands:

- init (and inittab)
- cron (and crontab)
- ps
- kill

In this chapter

This chapter contains the following topics.

cmnConfigRead

Purpose

cmnConfigRead is used by the installation process to read the configuration files.

cmnConfigRead reads the Convergent Charging Controller configuration file (**eserv.config**), specified by the Oracle_CONFIG_FILE environment variable and returns the value of `path`.

This can be used in commands to return the **eserv.config** specified path value.

Example:

```
FILENAME=`cmnConfigRead CCS.MyReport.filename
/IN/service_packages/CCS/tmp/MyReport.log`
```

This sets $FILENAME to the value of `CCS.MyReport.filename`. If `CCS.MyReport.filename` is not present or there is an error, $FILENAME defaults to **/IN/service_packages/CCS/tmp/MyReport.log**.

Startup

cmnConfigRead is started by the system and is not intended to be changed by the user.

cmnReceiveFiles

Purpose

cmnReceiveFiles collects EDRs from cmnPushFiles and writes them to the specified directory on the SMS.

Startup

cmnReceiveFiles is started by the following entry in **/etc/inetd.conf**:

```
smsoperFile     stream  tcp     nowait  smf_oper       /IN/service_packages/SMS/bin/
cmnReceiveFilesStartup.sh cmnReceiveFilesStartup.sh
```

Parameters

cmnReceiveFiles does not have any direct parameters or configuration. Most details are provided by cmnPushFiles with the EDR.

The port cmnReceiveFiles listens on is set in /etc/services in the following line:

```
smsoperFile     2028/tcp                       # cmnAddInetServicesEntry
```

Important: The port number must match the port specified by cmnPushFiles.

Failure

If cmnReceiveFiles fails, the EDRs stay on the SLC and are moved to the retry directory. For more information about this process, see *cmnPushFiles* (on page 150).

Output

cmnReceiveFiles writes the EDRs to the directory specified by cmnPushFiles.

smsAlarmDaemon

Purpose

The smsAlarmDaemon executable runs on all alarm-managed nodes in the SMS system, including the SMS node itself. The role of smsAlarmDaemon is to gather alarms from the following sources:

- Error messages log (**/var/adm/messages**)
- Oracle error log (**$ORACLE_BASE/admin/***SID***/bdump/alert_***SID***.log**)
- Sigtran stack logs (**/IN/service_packages/SLEE/stats**) [If installed]

On the SMS machine itself, the error messages are written directly into the SMF_ALARM_MESSAGE database table. When run on other nodes, replication is used to update the SMF_ALARM_MESSAGE table.

Alarm replication and buffering

smsAlarmDaemon allows only a limited number of alarms to be sent within a configured time period. Both the number of messages that can be sent within a time period and the length of each period can be configured from the command line.

If more messages arrive than are allowed through the filter, the remaining messages are buffered and sent later. The buffer size is limited but can hold a large number of messages. If it needs to make more space, it discards messages of the lowest severity (informational). The buffer also has an upper limit, ensuring that the daemons do not grow unchecked. This upper limit defaults to a maximum of 1000 messages and can be configured.

If more than one of the same alarm appears within the configured time, only one update request is sent.

Startup

In an unclustered install, this task is started by entry sms5 in the inittab, through the **/IN/service_packages/SMS/bin/smsAlarmDaemonSmsStartup.sh** shell script.

In a clustered install, this task is started by the clustering software, through the **/IN/service_packages/SMS/bin/smsAlarmDaemonCluster.sh** shell script.

Configuration

smsAlarmDaemon accepts the following command-line arguments.

Usage:

```
smsAlarmDaemon [-l seconds] [-h seconds] [-n number] [-m number] [-p] [-d] [-a path]
[-r node] [-u user/pass] [-f] [-i] [-g] [-c number] [-t seconds]
```

The available parameters are:

Parameter	Default	Description
`-a` *path*	Null	Propagate alarms from the specified Oracle alert log to the database. By default, smsAlarmDaemon does not propagate alarms from the Oracle alert log.
`-c` *number*	1	Commit Rate. The number of inserts before committing to the database.
`-d`	Sort messages	Disable sorting of messages in the buffer by severity. Specifically, messages are kept in the buffer and subsequently written into the SMF database, in the same sequence in which they are received.
`-f`	No filtering	Filtering. Delete duplicate alarms and increase the alarm count.
`-g`	Uses local time	GMT timezone. Use GMT instead of local time.
`-h` *seconds*	60	Heartbeat message. Will be forced to be greater or equal to time period (seconds).
`-i`	Use fuzzy matching	Filtering type. Use exact matching (rather than fuzzy matching). Indicates that duplicate matches should be performed on text only (that is, excluding digits). **Note:** Only valid when used in conjunction with **-f**.
`-l` *seconds*	2	Filter Period. Duration between linked-list checks (in seconds).
`-n` *number*	5	Filter Number. The number of alarm messages allowed within the time period. **Allowed values:** Integers
`-m` *number*	1000	Maximum number of alarm messages to buffer. **Allowed values:** Integers 1-1000000
`-p`	Do not drop messages	Drop low-priority messages when the buffer is full. Specifically, when **-m** *number* messages have been received but it is not yet time to write the buffer contents to the SMS database, low priority messages in the buffer are dropped in favor of higher-priority messages that may be received on its input stream.
`-r` *node*	Direct to the Oracle DB	Replication node. Specify the replication requester node.

Parameter	Default	Description
`-t` *`seconds`*	1	Commit interval. The maximum interval between database commits (in seconds).
`-u` *`user/pass`*	/	Use the supplied Oracle user/password pair.

Usage example

Here is an example of using smsAlarmDaemon:

```
smsAlarmDaemon -l 5 -h 30 -n 10 -m 2000 -p -d -a /volB/home/saich -r 750 -u
smf/smf -f -i -g -c 2 -t 2
```

- Filter Period (-l) = 5 seconds
- Heart beat (-h) = Yes every 30 seconds
- Filter Number (-n) = 10 each period
- Max number(-m) = 2000 records
- Drop low priority messages (-p) = true
- Sort messages by severity (-d) = false
- Oracle Alert Log location (-a) = /volB/home/saich
- Rep node (-r) = 750
- Oracle User (-u) = smf/smf
- Filtering (-f) = Multiple alarms combined
- Filtering type (-i) = Exact match
- GMT timezone (-g) = Yes
- Commit Rate (-c) = every 2 number of inserts
- Commit Interval (-t) = every 2 seconds if 2 records not reached

Failure

The smsAlarmDaemon on each alarm-managed node in the installation will by default generate a health-check alarm once per minute. These health check alarms will be relayed in the same fashion as all other alarms.

If these health check alarms are not received at the target destination, then the smsAlarmDaemon may have failed, and should be investigated.

Output

The smsAlarmDaemon writes error messages to the system messages file, and also writes additional output to **/IN/service_packages/SMS/tmp/smsAlarmDaemonSms.log**.

smsAlarmManager

Purpose

The smsAlarmManager runs on the SMS. The role of the smsAlarmManager is to:

- Match alarm instances to the correct alarm types
- Automatically time out alarms that have not been cleared
- record alarm instances that have no alarm type match

Startup

This task is started by entry efm1 in the inittab, through the **/IN/service_packages/EFM/bin/smsAlarmManagerStartup.sh** shell script.

The inittab entry will be similar to that shown below:

```
efm1:34:respawn:su - smf_oper -c "exec
/IN/service_packages/EFM/bin/smsAlarmManagerStartup.sh >> /IN
/service_packages/EFM/tmp/smsAlarmManager.log 2>&1" > /dev/null 2>&1 0<&1
```

Configuration

The smsAlarmManager accepts the following command line arguments.

Usage:

```
smsAlarmManager -a alarm_batch_size -c correlate_batch_size -o timeout_commit_rate -
p pending_timeout_length -r reload_defn_interval -s number -t timeout_check_interval
-u user/password
```

The available parameters are:

Parameter	Default	Description
`-a` *alarm_batch_size*	20	The number of alarms to attempt to find an ID for before carrying on to other tasks.
`-c` *correlate_batch_size*	20	The number of non-correlated CLEAR alarms to attempt to correlate against open alarms before carrying on to other tasks.
`-o` *timeout_commit_rate*	1000	The number of rows to update with automatic timeout before committing.
`-p` *pending_timeout_length*	280	The amount of time given to another node in the cluster before assuming that it has failed to generate an ALARM_TYPE_ID
`-r` *reload_defn_interval*	86400	interval (s) for reloading the alarm definitions for the DB. Interval between reloading the regular expressions from SMF_ALARM_DEFN and SMF_ALARM_IGNORE. This should only be needed after an install of new packages/patches, and also acts to keep the preferred cache current
`-s`	50000	interval (microseconds) to sleep for when no work to do
`-t` *timeout_check_interval*	300	Interval between checks for alarms that need to be closed with a timeout.
`-u` "*user/password*"	/	u "/" The username/password combination used to log into the database. The default value is sufficient if smsAlarmManager is executed from the smf_oper user account.

smsAlarmManager will respond to SIGHUP, to reread the regular expressions from the database.

Failure

The smsAlarmManager matches alarm instances with alarm types and updates the alarm instances with the extra information. Should any of the following occur the smsAlarmManager may have failed, and should be investigated.

- Alarms missing expected information
- Alarm clearances are not not being matched with the corresponding alarms
- Alarms not being automatically timed out

If the smsAlarmManager cannot match an alarm instance with an alarm type, it will save the alarm text into the SMF_ALARM_UNKNOWN database table.

Output

On startup the smsAlarmManager logs the following information:

```
smsAlarmManager startup.
    Alarm Batch Size = 20
    Correlate Batch Size = 20
    Pending Timeout Length = 280
    Timeout Check Interval = 300
    Reload Defn Interval = 86400
    Timeout Commit Rate = 1000
    Sleep Time (microseconds) for no Work = 50000
    Username/Password = /
Aug 30 15:31:07 smsAlarmManager(18347) NOTICE: smsAlarmManager started.
Cache successfully reloaded
```

smsAlarmRelay

Purpose

The smsAlarmRelay is responsible for implementing the *SNMP Agent* (on page 70). It runs continuously, polling the database to check for new entries written into the SMF_ALARM_MESSAGE table by the smsAlarmDaemon processes running on the various managed nodes which form the SMS-managed installation.

The information in the SMF_ALARM_MESSAGE is relayed to the destinations, as configured in the SMF_ALARM_HANDLER table using the Alarm Notification screens. For more information about how to configure alarm relay destinations, see the *Service Management System User's Guide*.

You can configure smsAlarmRelay to do the following:

- Send X.733 information with all forwarded alarms
- Check for SNMP requests (to resend alarms)
- Send version 3 (instead of version 1) SNMP traps

Startup

In an unclustered installation, this task is started by entry sms1 in the inittab, through the **/IN/service_packages/SMS/bin/smsAlarmRelayStartup.sh** shell script.

In a clustered installation, this task is started by the cluster software, through the **/IN/service_packages/SMS/bin/smsAlarmRelayCluster.sh** shell script.

Parameters

The `smsAlarmRelay` accepts the following command line arguments.

Usage:

```
smsAlarmRelay [-u <usr/pwd>] [-s <secs>] [-p] [-x] [-t] [-e]
```

Note: SNMP processing is not currently enabled by default.

The available parameters are:

`-u`

Syntax:	`-u` *`user/pwd`*
Description:	The Oracle user and password pair.
Type:	String
Optionality:	Optional (default used if not set).
Allowed:	
Default:	/
Notes:	
Example:	

`-s`

Syntax:	`-s` *`seconds`*
Description:	The number of seconds to sleep between database checks.
Type:	Integer
Optionality:	Optional (default used if not set).
Allowed:	
Default:	1
Notes:	
Example:	

`-p`

Syntax:	`-p`	
Description:	Whether to do SNMP processing or not.	
Type:	Boolean	
Optionality:	Optional (default used if not set).	
Allowed:	set	Use SNMP
	not set	Do not use SNMP
Default:	not set	
Notes:		
Example:		

`-x`

Syntax:	`-x`	
Description:	Whether to send SNMP traps in X.733 format or not.	
Type:	Boolean	
Optionality:	Optional (default used if not set).	
Allowed:	set	Send in X.733 format.
	not set	Do not send in X.733 format.
Default:	not set	
Notes:		
Example:		

`-t`

Syntax:	`-t`	
Description:	Whether to format the enterprise id with the severity.	
Type:	Boolean	
Optionality:	Optional (default used if not set).	
Allowed:	set	Insert severity into penultimate object of the extended enterprise id.
	not set	Do not format enterprise id with severity.
Default:	not set	
Notes:		
Example:		

`-e`

Syntax:	*`-e`*
Description:	Loads the EFM rules from the smf_alarm_relay_filter database table to provide alarm filtering.
Type:	Boolean
Optionality:	Optional.
Allowed:	Set or not set.
Default:	Not set

Resend Alarms

The smsAlarmRelay can be configured to listen for a request to resent all alarms above a certain alarm number. This is designed for use by an SNMP Manager that has been off line for a while and may have missed some alarm notifications.

To request a resend of alarms the relay application needs send an SNMP set-request using the format described in the **variables.mib** file.

The smsAlarmRelay will listen using the port number specified as the listenPort parameter in **snmp.cfg.** The alarmRelay keeps an internal count of the highest alarm number sent. When a valid SNMP set-request is received, the alarmRelay will take note of the number in the message and send all alarms with an alarm ID greater than this number.

Failure

The smsAlarmDaemon on each alarm-managed node in the installation will by default generate a health-check alarm once per minute. These health check alarms will be relayed in the same fashion as all other alarms.

If these health check alarms are not received at the target destination, then the smsAlarmRelay may have failed, and should be investigated.

Output

The **smsAlarmRelay.sh** writes error messages to the system messages file, and also writes additional output to **/IN/service_packages/SMS/tmp/smsAlarmRelay.log**.

The following table summarizes the information in the *Service Management System User's Guide*.

Destination	Field Content
SNMP	Host name of the target SNMP TRAPS recipient
FILE	Name of a file to which the daemon has write access
NFM	Host name of the NFM target.
Q3	Host name of the Q3 target.
SNMP	Host name of the SNMP target.
NORELAY	The field is empty, as the alarm is not forwarded to a target

Note: Setting the target to NORELAY will stop any other notification rules being actioned. Consequently, the NORELAY rules must be very specific. Otherwise an important alarm may accidentally be missed.

smsConfigDaemon

Purpose

smsConfigDaemon exists on both the source node (example, SMS) as well as the target node (example, SLC). It takes an optional parameter (`-m`) which decides its action.

When run with the `-m`, it monitors for changes to the master XML file (example, **esgConfigMaster.xml**). If it finds changes made to the master config file, smsConfigDaemon will call **smsSendConfig.sh**.

If the the `-m` parameter is missing, smsConfigDaemon monitors for changes to the derived **eserv.config** file (example, **eserv.config.derived**) on the target node, and calls **smsApplyConfig.sh** if it finds changes to the file.

About database connections

smsConfigDaemon connects to the database on a local or a remote SMS node by using the user credentials set in the following environment variables in **smsConfigVariables.sh**:

- SMP_DB_USER_NAME
- SMP_DB_PASSWORD
- SMP_DB_CONNECT_STRING

For connections to a:

- Local database, specify the username and password by setting the SMP_DB_USER_NAME and SMP_DB_PASSWORD variables. You can set only the user name in the SMP_DB_USER_NAME variable, if required.
- Remote database, specify the username and password by setting the SMP_DB_USER_NAME and SMP_DB_PASSWORD variables, and specify the SID of the remote database in the SMP_DB_CONNECT_STRING variable. You can set the SMP_DB_USER_NAME and the SMP_DB_CONNECT_STRING variables only, if required.
- Local or a remote database by using the Oracle wallet secure external password store, specify only the TNS connection string in the SMP_DB_CONNECT_STRING variable, where the connection string is the alias defined for the username and password credentials in the external password store. This alias can be either a TNS name or a service name from **tnsnames.ora**. The SMP_DB_CONNECT_STRING variable has the following format: "\@*connect_string*".

Note: If you set none of these variables, smsConfigDaemon connects to the database by using the default value of "/".

Startup

smsConfigDaemon is started by the script smsConfigDaemonScript. This process is driven by the system and is not intended to be changed by the user.

Configuration

For more information on the parameters used by smsConfigDaemon, see *smsConfigDaemonScript Configuration* (on page 104).

Failure

If the smsConfigDaemon fails, the secondary scripts, **smsSendConfig.sh** and **smsApplyConfig.sh** will fail to start and distribution of the updated configuration files is affected. Appropriate alarm messages are generated.

Output

The smsConfigDaemon and its sub-scripts write error messages to the system messages file, and also write additional output to **/IN/service_packages/SMS/tmp/smsConfigDaemonMaster.log** if they reside on the target node to **/IN/service_packages/SMS/tmp/smsConfigDaemonClient.log**.

smsConfigDaemonScript

Purpose

smsConfigDaemonScript is responsible for starting the smsConfigDaemon process. It also runs the **smsConfigVariables.sh** script which includes a set of configurable environment variables that are used by smsConfigDaemon and its helper scripts; for example, to set the username and password credentials for connecting to the Oracle database.

For more information about smsConfigDaemon, see *smsConfigDaemon* (on page 102).

Environment variables set in smsConfigVariables.sh

The **smsConfigVariables.sh** file is located in the following directory:

/IN/service_packages/SMS/bin

The following tables lists descriptions for the environment variables that you can configure in the **smsConfigVariables.sh** file and provides their default values.

Variable	Default Value	Description
SMP_DB_USER_NAME	None	The Oracle user that smsConfigDaemon uses to log in to the Oracle database.
SMP_DB_PASSWORD	None	The password for the Oracle user that smsConfigDaemon uses to log in to the Oracle database.
SCP_DB_USER_NAME	None	The Oracle user that the smsSignalConfigChange script uses to access sqlplus.
SCP_DB_PASSWORD	None	The password for the Oracle user that the smsSignalConfigChange script uses to access sqlplus.

Variable	Default Value	Description
SMP_DB_CONNECT_STRING	None	Any extra connect parameters that smsConfigDaemon requires to log in to the Oracle database.
DETECTION_PERIOD	10	The number of seconds between smsConfigDaemon change detection attempts.
RETRY_PERIOD	60	The number of seconds between smsConfigDaemon sendConfig retry attempts.
SLEEP_TIME	100	The number of milliseconds to sleep inside the smsConfigDaemon main loop.

Configuration

smsConfigDaemonScript sets the configurable parameters for smsConfigDaemon and its helper scripts.

The available parameters are:

Parameter	Default	Description
smp_db_user user/password		Oracle user/password for the SMF. **Example:** `smf/smf`
scp_db_user user/password		Oracle user/password for the smsSignalConfigChange script should use for sqlplus. **Example:** `scp/scp`
connect_string		Any extra connect parameters to be used by smsConfigDaemon
detection_period	10	Period (in seconds) after which smsConfigDaemon attempts to detect changes.
retry_period	60	Period (in seconds) after which smsConfigDaemon attempts to retry initiating sendConfig.
sleep_time	100	Period (in milliseconds) to wait inside smsConfigDaemon's main loop.
source_root	/IN/html/Configuration	Location where all the XML-driven config files and directories are stored. Not to be changed by the user.
master_xml_dir		Location of the master **config.xml** file. Not to be changed by the user.
master_config_file	esgConfigMaster	Name of the master configuration xml file.
master_config_file_full_path		Full path of the master configuration file monitored by the SMP config daemon (derived from the master config xml file).
archive_xml_dir		Location where the master config.xml files are archived to prior to modification.
derived_eserv_dir		Location of the derived eserv file.
pending_dir		Location where the config files from failed updates are held, pending a retry.
xml_convert_script		Location of the XML to **eserv.config** converter script.

Parameter	Default	Description
target_root	/IN/service_packages/Configuration	Location of the USP nodes where the **eserv.config** file is sent.
target_eserv_config_dir		Location on the USP nodes where the **eserv.config** file is pushed out to.
derived_eserv	**eserv.config**.derived	Name of the **eserv.config** file sent to the target nodes.
derived_config_file_full_path		Full path of the derived config file monitored by the SCP config daemon (derived from the **eserv.config** file sent to the target nodes)
management_interface_host	localhost	Location of the management interface.
management_interface_port		Port is the management interface listening on.

Note: It is not recommended to change the values of these parameters. All necessary configuration is done at installation time by the configuration script; this section exists for information only. Please contact Oracle support prior to attempting any modification to configuration data.

Startup

smsConfigDaemonScript is started by the system and is not intended to be changed by the user.

Failure

If smsConfigDaemonScript encounters problems, the smsConfigDaemon will fail to start and the updated **eserv.config** data will not be copied to the relevant platforms. Appropriate alarm messages are generated.

Output

The smsConfigDaemonScript writes error messages to the system messages file, and also writes additional output, when smsConfigDaemon has been started using the `-m` option, to the **/IN/service_packages/SMS/tmp/smsConfigDaemonMaster.log**.

If the `-m` option is not used, output will be written to **/IN/service_packages/SMS/tmp/smsConfigDaemonClient.log**.

smsCdrArchiver

Purpose

smsCdrArchiver performs a daily search of a specified input directory for CDR or EDR files to archive, and archives them to a file in a specified output directory. It also compresses and deletes old archive files according to the rules specified in the smsCdrArchiver configuration.

About archive file names

The name of the archive file output daily by smsCdrArchiver has the following format:

[*machineName*]+[*outputFileTag*]+[*_serviceType*]+*_Date*[-*HH*]+[*outputFileSuffix*]

Where

- *machineName* is the name of the machine that generated the data record. smsCdrArchiver prefixes the output file name with the machine name when you set the `useMachineName` parameter to true.
- *outputFileTag* is an identifying tag for the output file that you specify in the optional `outputFileTag` parameter.
- *serviceType* is the service type that generated the data record. smsCdrArchiver includes the service type in the output file name when you set the `useServiceType` parameter to true.
- *Date* is the date timestamp for the data record formatted as: YYYYMMDD.
- *HH* is the record hour that is appended to the *Date* value by using the following format: YYYYMMDD-HH, when you set the `UseRecordHour` parameter to true.
- outputFileSuffix is the suffix specified in the optional `outputFileSuffix` parameter that is appended to the output file name.

File name example

smsCdrArchiver has the following output file parameters configured in the **eserv.config** file:

```
    smsCdrArchiver = {
        ...
        outputFileTag = "ACS"
        outputFileSuffix = ".cdr"
        useRecordHour = true
        useMachineName = true
        useServiceType = true
        ...
        }
```

For machine name "telco-p-uas", service type "ACS", and timestamp "2014061512", the following output file would be created:

telco-p-uasACS_ACS_20140615-12.cdr

Startup

The smsCdrArchiver process is started by the **smsCdrArchiver.sh** script, that is located in the **/IN/service_packages/SMS/bin/** directory. The **smsCdrArchiver.sh** script runs in the crontab for the smf_oper user.

Configuration

You configure smsCdrArchiver in the SMS, smsCdrArchiver section of the **eserv.config** configuration file:

```
SMS = {

    smsCdrArchiver = {
        recordType = "CDR"
        inDir = "/cdr/processed"
        outDir = "/cdr/CDR-archive"
        outputFileSuffix = ".cdr"
        useRecordHour = true
        useMachineName = true
        useServiceType = true
        writeIndexFile = false

        useDateOutDirs = true
        prefixFileName2Data = false
        fileMatch = "telco-p-uas\*_ACS_"
        fileOwner = "smf_oper"
        compressionCommand = "GZIP"
        compressModTime = 2
        compressImmediately = false
        compressMinRunTime = 0
        deleteModTime = 31
        runCleanupHour = 03
        BFT = {

            exportBFTDataRecords = true
            exportBFTOutDir = "/cdr/export/BFT"

            exportBFTOutputFileSuffix = ""
            changeBFTOutputFileGroup = ""
            compressBFTDataRecords = true
            exportBFTKeepDays = 4
            ext5BFTHex2Dec = false
            zeroPadExt5Hex2Dec = 0
        }
    }
}
```

smsCdrArchiver parameters

The smsCdrArchiver section accepts the following parameters.

`recordType`

Syntax:	`recordType = "`*`str`*`"`
Description:	Defines the type of data records to archive. When `recordType` is set to: • CDR (for ACS Call Data Records), the ACS TCS (Time Call Start) tag is used to find the timestamp • EDR (for VWS Event Data Records), the VWS RECORD_DATE tag is used to find the date timestamp
Type:	String
Optionality:	Required
Allowed:	CDR, EDR
Default:	
Notes:	
Example:	`recordType = "CDR"`

`inDir`

Syntax:	`inDir = `*`"dir"`*
Description:	The directory that contains CDR or EDR input files.
Type:	String
Optionality:	Required
Allowed:	A vaild directory path and name.
Default:	
Notes:	ccsCdrArchiver will not search sub-directories of the specified directory for input files.
Example:	`inDir = "/cdr/processed"`

`outDir`

Syntax:	`outDir = `*`value`*
Description:	The output directory for the archived CDR or EDR file.
Type:	String
Optionality:	Required

Allowed:	A valid directory path and name
Default:	
Notes:	
Example:	`outDir = "/cdr/CDR-archive"`

outputFileTag

Syntax:	`outputFileTag = "`*`str`*`"`
Description:	An identifying tag for the output file, such as the name of the application that generated the data records. For example, for ACS CDRs set `outputFileTag` to "**`ACS`**".
Type:	String
Optionality:	Optional
Allowed:	
Default:	Not used
Notes:	
Example:	`outputFileTag = "ACS"`

outputFileSuffix

Syntax:	`outputFileSuffix = "`*`suffix`*`"`
Description:	The suffix to append to the name of the output file; for example, "**`.cdr`**" or ."**`edr`**"
Type:	String
Optionality:	Optional
Allowed:	
Default:	Not used
Notes:	
Example:	`outputFileSuffix = ".cdr"`

useRecordHour

Syntax:	`useRecordHour = true|false`
Description:	When set to true, the record hour is appended to the record date in the archive output file name by using the following format: YYYYMMDD-HH Where YYYYMMDD is the record date, and HH is the record hour.
Type:	Boolean
Optionality:	Optional
Allowed:	true, false
Default:	Not used
Notes:	
Example:	`useRecordHour = true`

useMachineName

Syntax:	`useMachineName = true|false`
Description:	When set to true, prefix the archive output file name with the name of the machine that generated the data record.
Type:	Boolean
Optionality:	Optional (default used if not set)
Allowed:	true, false

Default:	false
Notes:	The machine name can only be used in the output file name if the input file name has been prefixed with the machine name. This is the standard used by the cmnPushFiles process.
Example:	`useMachineName = true`

`useServiceType`

Syntax: `useServiceType = true|false`

Description: Include the data record service type tag in the output file name. The service type:

- For CDR records is specified in field 1
- For EDR records is specified in the CDR_TYPE field

Type:	Boolean
Optionality:	Optional (default used if not set)
Allowed:	true, false
Default:	false
Notes:	
Example:	`useServicetype = true`

`writeIndexFile`

Syntax: `writeIndexFile = true|false`

Description: When set, smsCdrArchciver writes an index file that links data record entries to the output file name. The name of the index file is *outputFilename***.idx**, where *outputFilename* is the archive output file name.

Index file entries have the following formats:

- CDR index file format: `Date Time CID CLI ServiceType(field 1)` [*`Data_SessionID`*]
- EDR index file format: `Date Time SEQUENCE_NUMBER CLI CDR_TYPE` [*`Data_SessionID`*]

Where:

- *`Data_SessionID`* is the ID for the data session
- CID in the CDR index file correlates to the VWS EDR SEQUENCE_NUMBER value (where applicable)

You use the index file to search, based on the fields listed above, for the identity of the archived output file containing the complete record.

Type:	Boolean
Optionality:	Optional (default used if not set)
Allowed:	true, false
Default:	false
Notes:	Using this option requires extra processing that can cause the smsCdrArchiver to run more slowly.
Example:	`writeIndexFile = true`

useDateOutDirs

Syntax:	`useDateOutDirs =` true|false
Description:	When set to true, smsCdrArchiver separates output files into date (YYYYMMDD) directories based on the TCS or RECORD_DATE values. The output files are written to the following location: *`outDir`*/*YYYYMMDD*/*`outFile`*
Type:	Boolean
Optionality:	Optional (default used if not set)
Allowed:	true, false
Default:	false
Notes:	Using this option requires extra processing that can cause the smsCdrArchiver to run more slowly.
Example:	`useDateOutDirs = true`

prefixFileName2Data

Syntax: `prefixFileName2Data = true|false`

Description: When set to true, smsCdrlArchiver prefixes the archived data record with the filename of the CDR or EDR record, by using the following format: *`original_filename`*:*`data_record_entry`*.

Where:

- *original_filename* is the name of the original file that contains the CDR or EDR record
- *data_record_entry* is the archived data record

You can use this option to identify the original filename in case of loading errors; for example, for CCS EDRs that are post-processed by the ccsCdrLoader.

Type:	Boolean
Optionality:	Optional (default used if not set)
Allowed:	true, false
Default:	false
Notes:	You should use this option for EDRs only.
Example:	`prefixFileName2Data = true`

fileMatch

Syntax: `fileMatch = "`*`str`*`"`

Description: Use to search for file names that match the prefix defined by the specified regular expression. You can define more than one prefix to match. The prefixes should be enclosed in double quotes "", and separated by white space.

You can include the following wild cards in prefix strings:

- * wild card at the end of each prefix string
- \ wild card to prevent shell expansion and unexpected results

Type:	String
Optionality:	Optional
Allowed:	
Default:	Not set
Notes:	
Example:	`filematch = "telco-p-uas\*_ACS_"`

`fileOwner`

Syntax:	`fileOwner = "`*`str`*`"`
Description:	When set, smsCdrArchiver locates only those files that are owned by the specified user.
Type:	String
Optionality:	Optional
Allowed:	
Default:	Not set
Notes:	
Example:	`fileOwner = "smf_oper"`

`compressionCommand`

Syntax:	`compressionCommand = "`*`str`*`"`
Description:	Specifies the compression utility to use for compressing old archive files.
Type:	String
Optionality:	Optional (default used if not set)
Allowed:	GZIP, BZIP2, PBZIP2
Default:	GZIP
Notes:	
Example:	`compressionCommand = GZIP`

`compressModTime`

ntax:	`compressModTime = `*`days`*
Description:	Specifies the minimum age in days for the archive file, based on its last modification time, before the archive file is compressed. Only files that are older than `compressModTime` will be compressed.
Type:	Integer
Optionality:	Optional (default used if not set)
Allowed:	
Default:	2 (48 hours)
Notes:	• If the `fileOwner` parameter is defined then only files that match the specified user are compressed • If a compressed file with the same name already exists then the existing compressed file is renamed to *originalFile_epochSec*.gz|bz2 • If you set this value to less than one (1) then compression is switched off
Example:	`compressModTime = 2`

`compressImmediately`

Syntax:	`compressImmediately = true|false`
Description:	When set to true, archive files are compressed immediately.
Type:	Boolean
Optionality:	Optional (default used if not set)
Allowed:	true, false
Default:	false
Notes:	
Example:	`compressImmediately = true`

compressMinRunTime

Syntax:	`compressMinRunTime = `*`seconds`*
Description:	The minimum number of seconds that archive file compression will run before pausing to check for new data records to archive. Setting this parameter helps to ensure that new input files are processed in a timely fashion.
Type:	Integer
Optionality:	Optional
Allowed:	An integer value between 60 (1 minute) and 3600 (1 hour). If you specify any other value then `compressMinRunTime` is switched off.
Default:	
Notes:	After compressing each file, smsCdrArchiver checks the threshold set by `compressMinRunTime`. If it has been passed, smsCdrArchiver pauses compression while new input files are archived, and then resumes the compression process.
Example:	`compressMinRunTime = 120`

deleteModTime

Syntax:	`deleteModTime = `*`days`*
Description:	The number of days to keep the archive files before they are deleted.
Type:	Integer
Optionality:	Optional
Allowed:	A valid integer. If `deleteModTime` is set to 0 (zero) or less, then archive file deletion is switched off, and no archive files are deleted.
Default:	
Notes:	If the `useDateOutDirs` parameter is set to true, then smsCdrArchiver deletes the date directories that are older than the `deleteModTime` value. Otherwise the number of days that an archive file is kept is based its last modification date.
Example:	`deleteModTime = 31`

runCleanupHour

Syntax:	`runCleanupHour = `*`int`*
Description:	Specifies the hour of the day to perform the smsCdrArchiver clean-up function to compress and delete the old archive files. The hour is based on the local machine time (specified by the TZ variable in the **sms.jnlp** file). You must ensure that the value of `runCleanupHour` matches the crontab hour (specified in field 2), unless the crontab hour is set to * .
Type:	Integer
Optionality:	Optional (default used if not set)
Allowed:	An integer between 0 and 23
Default:	
Notes:	The clean-up of old archive files is performed once per day. You can force smsCdrArchiver to rerun the clean-up by deleting the following file: **outDir/.smscdrArchiver.dayofYear.def**
Example:	`runCleanupHour = 03`

Billing Failure Treatment CDR parameters

The Billing Failure Treatment (BFT) parameters define rules for exporting BFT data records to a special directory for BFT post processing by the billing server.

Note: The BFT parameters apply only to ACS CDRs, where the `recordType` field is set to "CDR".

You configure BFT parameters in the smsCdrArchiver, BFT section of the **eserv.config** file by using the following syntax:

```
smsCdrArchiver = {
    BFT = {

        exportBFTDataRecords = true
        exportBFTOutDir = "/cdr/export/BFT"

        exportBFTOutputFileSuffix = ""
        changeBFTOutputFileGroup = ""
        compressBFTDataRecords = true
        exportBFTKeepDays = 4
        ext5BFTHex2Dec = false
        zeroPadExt5Hex2Dec = 0
    }
}
```

`exportBFTDataRecords`

Syntax:	`exportBFTDataRecords = true|false`
Description:	Enables BFT CDRs to be exported to the directory specified by the `exportBFTOutDir` parameter.
Type:	Boolean
Optionality:	Optional (default used if not set)
Allowed:	true, false
Default:	false
Notes:	
Example:	`exportBFTDataRecords = true`

`exportBFTOutDir`

Syntax:	`exportBFTOutDir = "`*`str`*`"`
Description:	The directory path for the directory to which to export BFT CDRs.
Type:	String
Optionality:	Optional
Allowed:	
Default:	
Notes:	
Example:	`exportBFTOutDir = "/cdr/export/BFT"`

`exportBFTOutputFileSuffix`

Syntax:	`exportBFTOutputFileSuffix = "`*`str`*`"`
Description:	Specifies the suffix for the BFT output file. If this is unset, then the original input file name is used.
Type:	String
Optionality:	Optional

Allowed:	
Default:	
Notes:	White spaces are replaced by the _ (underscore) character.
Example:	`exportBFTOutputFileSuffix = ""`

changeBFTOutputFileGroup

Syntax:	`changeBFTOutputFileGroup = "`*`str`*`"`
Description:	Sets the group file permissions for the output file; for example, to change group read/write access to allow third parties to collect BFT CDRs for post processing.
Type:	String
Optionality:	Optional (default used if not set)
Allowed:	The group file permissions must be valid for the user running the smsCdrArchiver script.
Default:	
Notes:	If the group is invalid, or left undefined, then the group file permissions are not changed.
Example:	`changeBFTOutputFileGroup = ""`

compressBFTDataRecords

Syntax:	`compressBFTDataRecords = true|false`
Description:	Set to true to compress the BFT oputput files as they are written. If set to false, then the `compressModTime` parameter does not take the BFT CDR files into account and no further compression will be done.
Type:	Boolean
Optionality:	Optional (default used if not set)
Allowed:	true, false
Default:	true
Notes:	The compression utility used is defined by the `compressionCommand` parameter.
Example:	`compressBFTDataRecords = true`

exportBFTKeepDays

Syntax:	`exportBFTKeepDays = `*`int`*
Description:	The number of days to keep the exported BFT CDRs before they are deleted.
Type:	Integer
Optionality:	Optional (default used if not set)
Allowed:	
Default:	1 - The default value of one will be used if you specify a value that is less than one.
Notes:	
Example:	`exportBFTKeepDays = 4`

ext5BFTHex2Dec

Syntax:	`ext5BFTHex2Dec = true|false`
Description:	Whether the ACS CDR EXT5 field is written as a hexidecimal value, such as **`EXT5=0000000A`**, or whether it is converted to a decimal value, such as **`EXT5=10`**, for the BFT post-processing tools.
Type:	Boolean
Optionality:	Optional (default used if not set)

Allowed:	• true (use decimal values) • false (use hex values)
Default:	false
Notes:	
Example:	ext5BFTHex2Dec `= false`

`zeroPadExt5Hext2Dec`

Syntax:	`zeroPadExt5Hext2Dec =` *`int`*
Description:	The number of leading zeros to use when padding the converted EXT5 decimal number (if `ext5BFTHex2Dec` is set to true). Set to 0 (zero) or 1 (one) for no leading zero padding.
Type:	Integer
Optionality:	Optional (default used if not set)
Allowed:	
Default:	0
Notes:	If `zeroPadExt5Hext2Dec` is set to a negative number then the decimal number will be padded so that it is the same length as the original hex number. For example, the hex number: **`0000000A`** is converted to the decimal number: **`00000010`**.
Example:	`zeroPadExt5Hext2Dec = 0`

smsCdrProcess.sh

Purpose

smsCdrProcess.sh performs basic EDR processing and archiving. **smsCdrProcess.sh** runs the smsProcessCdr binary with specified command line parameters. One output EDR file is created for each input EDR file.

smsCdrProcess.sh can also be configured to prevent processing the EDR.

For more information about how smsProcessCdr processes EDRs, see *smsProcessCdr* (on page 198).

EDR format

The format of the records in the EDR file are specific to the application which generates them. The most commonly used EDR format processed by this mechanism is the ACS "Pipe Tag LF" format, which uses TAG=VALUE pairs separated by the "|" character. Records are line field delimited.

For more information about this format, see the *ACS Technical Guide*.

Startup

This task is run in the crontab for smf_oper, by default at 1:00 am system clock time. It is scheduled as the **/IN/service_packages/SMS/bin/smsCdrProcess.sh** script.

The script runs the smsProcessCdr process with set parameters.

Configuration

The following command in the **smsProcessCdr.sh** prevents the EDR from being processed and copies it directly to the output directory.

Example Command: `$BINDIR/smsProcessCdr -d $CDRDIR -D $OUTDIR -s $INSFX -S $OUTSFX`

To process EDRs, use the following command instead:

Example Command: `$BINDIR/smsProcessCdr -t $OUTFMT -d $CDRDIR -D $OUTDIR -s $INSFX -S $OUTSFX`

Failure

If the process is not running, EDR files will build up in the **/IN/service_packages/SMS/cdr/received** directory.

The filesystem usage will rise above standard operational levels.

Output

The **smsCdrProcess.sh** writes error messages to the system messages file, and also writes additional output to **/IN/service_packages/SMS/tmp/smsCdrProcess.sh.log**.

smsDbCleanup.sh

Purpose

This task executes SQL statements to delete old data from the following tables.

SQL Statement	Data deleted
SMF_AUDIT	Audit trail of database changes.
SMF_STATISTICS	Application bulk usage counters.
SMF_TEMPLATE	Operator permissions templates.
SMF_ALARM_MESSAGE	System messages.

Startup

This task is run in the crontab for smf_oper, by default at 1:00 am system clock time. It is a shell script, specifically **/IN/service_packages/SMS/bin/smsDbCleanup.sh**.

Parameters

The decision on when to delete data is determined according to various parameters configured in **eserv.config**. These values can be changed by the usual **eserv.config** editing method, subject to database sizing limitations and availability of space for additional historical data.

The default parameters are:

Parameter	Default	Description
`alarmAge`	7	Delete records older than this number of days. Refers to the actual age of the alarm, and controls the deletion of all alarms of a certain age, regardless of whether they are noted (closed).
`alarmMax`	100000	Maximum number of records to keep. After this value is reached, smsDbCleanup.sh delete records.
`alarmNotedAge`	3	Controls the deletion of noted (closed) alarms.
`auditAge`	7	Delete records older than this number of days.
`commit`	100	Number of statistic records to delete before committing the deletions.
`statsAge`	30	Delete records older than this number of days.

Parameter	Default	Description
`templateAge`	1	Delete templates not allocated to any operator older than this number of days.
`unknownMax`	5000	Maximum number of alarms to keep in table smf_alarm_unknown. After this value is reached, new additions cause oldest to be deleted from the table.

Failure

If the process is not running, old data will not be purged from the database. The database may reach maximum size, and inserts may fail.

Output

The **smsDbCleanup.sh** writes error messages to the system messages file, and also writes additional output to **/IN/service_packages/SMS/tmp/smsDbCleanup.sh.log**.

smsLogCleaner

Purpose

smsLogCleaner archives the following types of log files:

- Convergent Charging Controller process log files (/IN/service_packages/*Product*/tmp/*Process*.log)
- System log files (syslog)

For more information, see *System Administrator's Guide*.

Startup

This task is run in the crontab for smf_oper. By default, it runs at 30 minutes past each hour. It is run via the shell script:

```
/IN/service_packages/SMS/bin/smsLogCleanerStartup.sh
```

Parameters

The smsLogCleaner supports the following command-line options:

Usage:

```
smsLogCleaner -c configuration_file -d days -s storage_file [-h]
```

The available parameters are:

Parameter	Default	Description
-c *configuration_ file*	**logjob.conf**	The name of the configuration file to use.
-d days	7	How often to clean archive in days.
-s *storage_file*	**storage.txt**	The name of the storage file to use.
-h		Provides help information.

At installation time, the cronjob is configured to execute by default with the following command line parameters:

`-c` **/IN/service_packages/SMS/etc/logjob.conf**

`-s` **/IN/service_packages/SMS/tmp/sms_storage.txt**

`-d 7`

An operator may change these values, subject to disk storage availability and site-specific archiving policies.

Failure

If the process is not running, log files in the following directory will accumulate in size and age beyond the expected values.

```
/IN/service_packages/SMS/tmp
```

Output

The smsLogCleaner run by smf_oper writes error messages to the system messages file, and also writes additional output to **/IN/service_packages/SMS/tmp/smsLogCleaner.log**.

logjob.conf

The logjob.conf configuration file has the following format:

```
log <file> age <hrs> size <size> arcdir <dir> logonce zip <size>
```

The available parameters are:

Parameter	Description
log <file>	The full directory path and name of the file to be cleaned. You can include the '*' wildcard in the file name if required. **Example:** `log /IN/service_packages/SMS/tmp/smsNamingServer.log` **Tip:** Most processes and tools document where their output is written to in their Output topic.
age <hrs>	Sets the minimum age in hours for the log file before it will be cleaned. You must set either this parameter or the size parameter. If both parameters are set, then the log file is cleaned if either condition is met. **Example:** `age 100`
size <size>	Sets the minimum size for the logfile before it will be cleaned. You must set either this parameter or the age parameter. If both parameters are set, then the log file is cleaned if either condition is met. **Examples:** `size 60K`, or `size 60M`
arcdir <dir>	The directory to use to store the old log file. If this parameter is not specified, then the log file is deleted. **Example:** `arcdir /IN/service_packages/SMS/tmp/archive`
logonce	Only specify this parameter if you just want to keep one archived version of the log file.
zip <size>	Automatically compress log files that exceed the specified size. **Example**: `zip 100M`

smsMergeDaemon

Purpose

The smsMergeDaemon monitors the connections between the SMS and SLCs via a heartbeat. The smsMergeDaemon will initiate a startMerge to resyncronise the SMS and SLCs where:

- the infMaster on the disconnected SLC reports that it has received updates that would have normally gone to the SMS or it has an updateLoader or updateRequester pointing to it, and
- the heartbeat to the SMS and SLC have been stable for a period.

For more information about the startMerge process, see *startMerge* (on page 207).

Startup

This task is started by entry sms9 in the inittab, via the shell script:

```
/IN/service_packages/SMS/bin/smsMergeDaemonStartup.sh
```

Note: smsMergeDaemon is not used in a clustered install.

Parameters

The smsMergeDaemon accepts the following command line arguments.

Usage:

```
smsMergeDaemon -nodeid
```

The available parameters are:

Parameter	Default	Description
-nodeid	1000	The node number of the smsMergeDaemon.

The rest of the configuration details are taken from *replication.def* (on page 30). Relevant parameters include:

- HB PERIOD
- LONG TIMEOUT
- MAX_ROUNDTRIP 3
- MAX_CONNECTION_TIME 100000000
- MERGE_INTERVAL 600
- REP_PATH "/IN/service_packages/SMS/etc/replication.config"
- SMS_PORT 7
- TICK_TIME 1000

Failure

If the smsMergeDaemon's connection to the smsMaster is lost, it will exit.

Output

The smsMergeDaemon.sh writes error messages to the system messages file, and also writes additional output to **/IN/service_packages/SMS/tmp/smsMergeDaemon.log**.

smsMaster

Purpose

The smsMaster is the central correlation point for the replication system.

smsMaster:

- Sends notifications of updates to remote updateLoaders to be loaded into secondary databases.
- Accepts update requests from remote systems that wish to change the master database (including the smsStatsDaemon, RequesterIF and smsAlarmDaemon).
- Correlates full resynchronization with remote databases, and communicates with inferior masters which can assume some smsMaster functions in the case of a platform or network failure.

Startup

This task is started by entry sms7 in the inittab, through the shell script:

```
/IN/service_packages/SMS/bin/smsMasterStartup.sh
```

Configuration

The smsMaster supports the following command-line options:

Usage:

```
smsMaster -maxpending int
```

The available parameters are:

`-maxpending`

Syntax:	`-maxpending` *`int`*
Description:	The size of pending request queue.
Type:	Integer
Optionality:	Optional (default used if not set).
Allowed:	
Default:	10000
Notes:	
Example:	

Failure

Remote replication nodes such as update loaders, updated requesters, inferior masters will generate alarms indicating connection failure to the smsMaster.

Output

The smsMaster writes error messages to the system messages file, and also writes additional output to **/IN/service_packages/SMS/tmp/smsMaster.log**.

smsNamingServer

Introduction

The smsNamingServer listens for IORs being exported from CORBA processes, and stores them in the database in the IORS table owned by Oracle user SMF. It also serves requests to read IOR strings from the database.

This functionality is required to support processes that wish to store/retrieve IOR strings, but which do not have Oracle access to the SMF database instance, for security or licensing reasons.

Startup

In an unclustered installation, this task is started by entry sms2 in the inittab, through the shell script:

```
/IN/service_packages/SMS/bin/smsNamingServerStartup.sh
```

In a clustered installation this task is started by the cluster software, through the shell script:

```
/IN/service_packages/SMS/bin/smsNamingServerCluster.sh
```

Parameters

The smsNamingServer supports the following command-line options:

Usage:

```
smsNamingServer [-u usr/pwd] [-p port]
```

The available parameters are:

`-u`

Syntax:	`-u` *usr/pwd*
Description:	The Oracle user and password pair.
Type:	String
Optionality:	Optional (default used if not set)
Allowed:	
Default:	/
Notes:	
Example:	`-u /`

`-p`

Syntax:	`-p` *port*
Description:	The port number on which to listen for requests.
Type:	Integer
Optionality:	Optional (default used if not set)
Allowed:	
Default:	7362
Notes:	
Example:	`-p 7362`

Failure

If the smsNamingServer fails, then processes attempting to access the specified port will not be able to access the service, and should report an error indicating this.

Output

The smsNamingServer writes error messages to the system messages file, and also writes additional output to **/IN/service_packages/SMS/tmp/smsNamingServer.log**.

smsReportsDaemon

Purpose

smsReportsDaemon is a CORBA server process that generates reports on demand.

When the SMS user interface (UI) Reports function requests a report, smsReportsDaemon:

- Returns the report output filename
- Writes the report output to a specified directory on the SMS

The SMS UI Reports function then displays the report. For more information about the SMS Reports function, see *Service Management System User's Guide*.

Startup

In an unclustered installation, smsReportsDaemon is started by entry sms3 in the inittab, via the shell script:

```
/IN/service_packages/SMS/bin/smsReportsDaemonStartup.sh
```

In a clustered installation, smsReportsDaemon is started by the cluster software, via the shell script:

```
/IN/service_packages/SMS/bin/smsReportsDaemonCluster.sh
```

Parameters

smsReportsDaemon supports the following command-line options.

Usage:

```
smsReportsDaemon [-h host] [-p port] [-i dir] [-o dir] [-f dir] [-u user/password]
[-t host] [-s port] [-z timezone] [-m num]
```

The available parameters are:

Parameter	Default	Description
-h *host*	Value returned by gethostname	smsNamingServer hostname. Allowed value type: ASCII String
-p *port*	7362	smsNamingServer port number. Allowed value type: Number
-i *dir*	**/IN/service_packages/SMS/input**	Report scripts/binaries input directory. Allowed value type: ASCII String
-o *dir*	**/IN/service_packages/SMS/output**	Generated report output directory. Allowed value type: ASCII String
-f *dir*	**/IN/service_packages/SMS/input**	The **setTZ.sql** file directory. By default, the file is located in **/IN/service_packages/SMS/input.**
-u *user/password*		Oracle SMF database username and password. Allowed value type: ASCII String

Parameter	Default	Description
-t *host*	Default determined by CORBA	CORBA transport layer hostname.
-s *port*	Default determined by CORBA	CORBA transport layer port number. This parameter is ignored if the CorbaServices section is present in the **eserv.config** configuration file. For more information, see *Configuring Connections for CORBA Services* (on page 76).
-z *timezone*		Timezone in which the smsReportsDaemon SQL queries are run generating the report output.
-m *num*	2	Maximum number of concurrent reports per node.

Failure

If smsReportsDaemon fails, you will not be able to generate reports.

Output

smsReportsDaemon writes error messages to the system messages file, and writes additional output to **/IN/service_packages/SMS/tmp/smsReportsDaemon.log**.

smsReportsDaemon writes report output to subdirectories of the specified output directory (by default, **/IN/service_packages/SMS/output**). The subdirectory depends on the application and category defined for the report:

/IN/service_packages/SMS/output/*Application***/***Category*

The report output filename is in the format:

`YYMMDDHHmmss.9_random_characters`**.txt**

Interactive reports

smsReportsDaemon generates on-demand reports.

Reports are defined through SQL commands, shell scripts, or compiled executable programs. Additional reports can be created and made available for on-demand and scheduled generation as a post-installation manual function. For more information, see *Reports* (on page 209).

At startup, smsReportsDaemon publishes its IOR string via the smsNamingServer. If not specified, the IP port number on which the CORBA service is provided will be determined by the CORBA framework. In most installations, a firewall is used to protect the SMS host, and hence the CORBA service port must be fixed. Use the `-s` parameter for this purpose.

smsReportScheduler

Purpose

The smsReportScheduler monitors the database table SMF_REPORT_SCHEDULE for entries inserted via the SMS Java screens.

smsReportScheduler sleeps until the next report is due to be executed. The output of the report is optionally copied to a specified directory, spooled to a specified printer, or sent to a specified email address. For more information about how to schedule reports which will be performed periodically and how to configure the report destination, see the *Service Management System User's Guide*.

Startup

In an unclustered installation, this task is started by entry sms4 in the inittab, via the shell script:

```
/IN/service_packages/SMS/bin/smsReportSchedulerStartup.sh
```

In a clustered installation this task is started by the cluster software, via the shell script:

```
/IN/service_packages/SMS/bin/smsReportSchedulerCluster.sh
```

Parameters

The smsReportScheduler supports the following command-line options:

Usage:

```
smsReportScheduler [-i dir] [-o dir] [-u usr/pwd] [-v] [-z timezone]
```

The available parameters are:

-i dir

Syntax:	`-i` *dir*
Description:	The input directory for report generation scripts/binaries dir.
Type:	String
Optionality:	Optional (default used if not set)
Allowed:	
Default:	**/IN/service_packages/SMS/input**
Notes:	
Example:	

-o dir

Syntax:	`-o` *dir*
Description:	The output directory for report generation.
Type:	String
Optionality:	Optional (default used if not set)
Allowed:	
Default:	/IN/service_packages/SMS/output
Notes:	Report may provide a default output directory which overrides smsReportDaemon's default.
Example:	

-u usr/pwd

Syntax:	`-u` *usr/pwd*
Description:	The userid and password for oracle login string.
Type:	String
Optionality:	Optional (default used if not set)
Allowed:	
Default:	/
Notes:	
Example:	

`-v`

Syntax:	`-v`
Description:	What level of information to output.
Type:	Boolean
Optionality:	Optional (default used if not set)
Allowed:	set — Print additional information. not set — Only print the standard level of information.
Default:	not set
Notes:	
Example:	

`-z timezone`

Syntax:	`-z` *`timezone`*
Description:	The timezone in which to schedule the report.
Type:	String
Optionality:	Optional (default used if not set)
Allowed:	Java supported timezone
Default:	GMT
Notes:	For a full list of Java supported timezones see ACS Technical Guide - Appendix TimeZones.
Example:	`-z "EST"`

Failure

In the case of failure, the scheduled report will not appear at the specified destination, or may contain incorrect or missing output.

Output

The smsReportScheduler writes error messages to the system messages file, and also writes additional output to **/IN/service_packages/SMS/tmp/smsReportScheduler.log**.

Unix utilities

The table below lists the Unix utilities required for scheduling.

Unix Binary Required	Description	Location Expected
mailto	E-mail agent used by report generation component to send emails.	/usr/bin/mailto
sendmail (or equivalent delivery agent)	E-mail delivery agent.	daemon started at boot time.
lpr	Printing utility.	/usr/ucb/lpr

Note: E-mail sending/receiving/delivery agent requires all local e-mail user names to be under 13 characters. For local e-mail user names longer than 13 characters, mailto and sendmail will not function properly.

smsReportCleanupStartup.sh

Purpose

The Reports cleaner looks for output from ad-hoc and scheduled reports generated by the smsReportsDaemon and the smsReportScheduler.

It deletes files that are older than a specified age.

Startup

This task is run in the crontab for smf_oper. By default it runs at 2:00 am system time. It is scheduled as the following script:

```
/IN/service_packages/SMS/bin/smsReportsCleanerStartup.sh
```

Parameters

The command inside the script contains a command line parameter specifying the cleanup age of report output files. By default this is seven days. Report output files older than this age are deleted.

An operator may change this value, subject to disk storage availability and site-specific archiving policies.

Failure

If the process is not running, reports files in the following directory will accumulate in size and age beyond the expected values.

/IN/service_packages/SMS/output

Output

The smsReportsCleaner run by smf_oper writes error messages to the system messages file, and also writes additional output to:

/IN/service_packages/SMS/tmp/smsReportsCleanerStartup.sh.log

smsStatsDaemon

Description

smsStatsDaemon can be run as a background process on an SMS, SLC, and also on other IPs such as Voucher and Wallet Servers.

For more information about smsStatsDaemon, see the discussion about the *smsStatsDaemon* (on page 160) background process on the SLC node.

smsStatisticsWriter

Purpose

The smsStatisticsWriter is responsible for collecting statistical data provided by the smsStatDaemons and writing it to files, either for standalone statistics, or for groups of statistics associated with an event.

smsStatisticsWriter structure

Here is an example structure of the smsStatisticsWriter.config file.

```
smsStatisticsWriter = {
    tempDir = "/IN/service_packages/SMS/tmp/smsStatisticsWriter"
    outDir = "/IN/service_packages/SMS/logs/smsStatisticsWriter"
    outDirType = 'FLAT'
    outDirExpectedFiles = 65536
    outDirBucketSize = 10
    outFileName = "smsStatisticsWriter"
    maxFileSize = 100
    maxFileOpenTime = 3600
    statsDaemonRestartDelay = 10
    statsDaemonStartupTime = 5
    scanInterval = 100000
    replicationAllowance = 60
    endOfEventTolerance = 5
    resetInterval = {
        days = 1
        hours = 0
        minutes = 0
        seconds = 0
    }

    Events = [
        {
            eventName = "EventGood"
            resetAllEventStatisticsOnStartup = true
            eventResetBaseTime = "20110511130000"
            resetInterval = {
                days = 0
                hours = 0
                minutes = 4
                seconds = 0
        }
        eventStartDateTime = "20100728000000"
        eventEndDateTime = "20101231000000"
        eventWritePeriod = 30
        Statistics = [
            {
                applicationName = "TELEVOTING"
                statisticName = "Stat1"
            }

            {
                applicationName ="TELEVOTING"
                statisticName = Statn
            }
            etc
        ]

        {
            eventName = "Event2"
            parameters for event...
        }

        {
            eventName = "Eventn"
            parameters for event...
        }

    ]
```

```
        Statistics = [
            {
                StatisticName = "Stat2"
                resetStatisticOnStartup = false
                statWritePeriod = 0
                statResetBaseTime = "20110511130000"
                resetInterval = {
                    days = 0
                    hours = 0
                    minutes = 5
                    seconds = 0
                }
            }
            {
                statisticName = "StatN"
                parameter for statistic...
            }
            Etc
        ]
```

smsStatisticsWriter parameters

The smsStatisticsWriter section accepts the following parameters.

`tempDir`

Syntax:	`tempDir = "`*`dir`*`"`
Description:	The temporary directory for writing intermediate statistics files to.
Type:	String
Optionality:	Optional (default used if not set).
Allowed:	
Default:	"/IN/service_packages/SMS/tmp/smsStatisticsWriter"
Notes:	
Example:	`tempDir = "/IN/service_packages/SMS/tmp/smsStatisticsWriter"`

`outDir`

Syntax:	`outDir = "`*`dir`*`"`
Description:	The base directory for storing completed statistics files.
Type:	String
Optionality:	Optional (default used if not set).
Allowed:	
Default:	"/IN/service_packages/SMS/logs/smsStatisticsWriter"
Notes:	
Example:	`outDir = "/IN/service_packages/SMS/logs/smsStatisticsWriter"`

`outDirType`

Syntax:	`outDirType: = "`*`type`*`"`
Description:	File store type.
Type:	String
Optionality:	Optional (default used if not set).
Allowed:	HASH or FLAT

Default:	"FLAT"
Notes:	
Example:	`outDirType: = "FLAT"`

outDirExpectedFiles

Syntax:	`outDirExpectedFiles =` *`num`*
Description:	The maximum number of files in outDir when outDirType is HASH.
Type:	Integer
Optionality:	Optional (default used if not set).
Allowed:	
Default:	65536
Notes:	
Example:	`outDirExpectedFiles = 65536`

outDirBucketSize

Syntax:	`outDirBucketSize =` *`size`*
Description:	The maximun number of files in any leaf directory when outDirType is HASH.
Type:	Integer
Optionality:	Optional (default used if not set).
Allowed:	
Default:	10
Notes:	
Example:	`outDirBucketSize = 10`

outFileName

Syntax:	`outFileName = "`*`name`*`"`
Description:	The base of the statistics file name.
Type:	String
Optionality:	Optional (default used if not set).
Allowed:	
Default:	"smsStatisticsWriter"
Notes:	
Example:	`outFileName = "smsStatisticsWriter"`

maxFileSize

Syntax:	`maxFileSize =` *`size`*
Description:	The maximum size of a statistics file (kilobytes).
Type:	Integer
Optionality:	Optional (default used if not set).
Allowed:	
Default:	100
Notes:	
Example:	`maxFileSize = 100`

`maxFileOpenTime`

Syntax:	`maxFileOpenTime = `*`sec`*
Description:	The maximum time a statistics file can be kept open (seconds).
Type:	Integer
Optionality:	Optional (default used if not set).
Allowed:	
Default:	3600
Notes:	
Example:	`maxFileOpenTime = 3600`

`statsDaemonRestartDelay`

Syntax:	`statsDaemonRestartDelay = `*`sec`*
Description:	The length of time to allow updated statistics to be replicated to the SLC nodes before restarting the smsStatsDaemon processes on those nodes. (seconds).
Type:	Integer
Optionality:	Optional (default used if not set).
Allowed:	
Default:	10
Notes:	
Example:	`statsDaemonRestartDelay = 10`

`statsDaemonStartupTime`

Syntax:	`statsDaemonStartupTime = <seconds>`
Description:	The length of time (in seconds) to allow for **smsStatsDaemon** to restart and initialize after **startUp** request issued.
Type:	Integer
Optionality:	Optional (default used if not set).
Allowed:	
Default:	5
Notes:	
Example:	`statsDaemonStartupTime = 8`

`scanInterval`

Syntax:	`scanInterval = `*msec*
Description:	Length of time between scans of the statistics (microseconds).
Type:	Integer
Optionality:	Optional (default used if not set).
Allowed:	
Default:	100000
Notes:	
Example:	`scanInterval = 100000`

`replicationAllowance`

Syntax:	`replicationAllowance = <seconds>`
Description:	The time (in seconds) to allow for stats to be replicated to SMS from SLC.

Type:	Integer
Optionality:	Optional (default used if not set).
Allowed:	
Default:	60
Notes:	For best results this value should be less than any of the reporting periods of statistics included in event configuration.
Example:	`replicationAllowance = 40`

Events

Event configuration parameters may or may not have default values, where defaults are defined that parameter may be omitted, for all others the parameter is required. There may be multiple events configured.

Each event accepts the following parameters.

```
Events = [
    {
        eventName = "EventGood"
        resetAllEventStatisticsOnStartup = true
        eventResetBaseTime = "20110511130000"
        resetInterval = {
            days = 0
            hours = 0
            minutes = 4
            seconds = 0
    }
    eventStartDateTime = "20100728000000"
    eventEndDateTime = "20101231000000"
    eventWritePeriod = 30
    Statistics = [
        {
            applicationName = "TELEVOTING"
            statisticName = "Stat1"
}
```

eventName

Syntax:	`eventName = "`*`name`*`"`
Description:	The name of the event.
Type:	String
Optionality:	Required
Allowed:	
Default:	no default
Notes:	
Example:	`eventName = "eventGood"`

resetAllEventStatisticsOnStartup

Syntax:	`resetAllEventStatisticsOnStartup = `*`true|false`*
Description:	Reset all statistics defined for the event on start up.
Type:	Boolean
Optionality:	Optional (default used when omitted)
Allowed:	• true, or • false

Default:	true
Notes:	If true, individual statistic resetStatisticOnStatup config is overidden.
Example:	`resetAllEventStatisticsOnStartup = false`

eventResetBaseTime

Syntax:	`eventResetBaseTime = "<date and time>"`
Description:	Sets a specific date and time base to calculate resets for the statistics configured for this event.
Type:	String
Optionality:	Optional (default used if not set).
Allowed:	
Default:	"20100101000000" - (00:00:00 on Jan 1 2010)
Notes:	Date and time format is yyyymmddHHMMSS The event **resetInterval** parameter timing starts from this parameter's date and time.
Example:	`eventResetBaseTime = "20110511130000"`

resetInterval

Syntax:	`resetInterval = {<days>,<hours>,<minutes>,<seconds>}`
Description:	The interval between periodic resets of all statistics configured for the event.
Type:	Parameter list
Optionality:	Optional (default used if missing)
Allowed:	
Default:	Global resetInterval values are used.
Notes:	If non zero, the reset period configured for individual statistics will be overridden.

Example:

```
resetInterval = {
    days = 0
    hours = 0
    minutes = 4
    seconds = 0
}
```

eventStartDateTime

Syntax:	`eventStartDateTime = "`*`dateandtime`*`"`
Description:	The event start time.
Type:	String
Optionality:	Required
Allowed:	Expected format yyyymmddHHMMSS
Default:	no default
Notes:	Statistics will only be collected for the event between the start and end times.
Example:	`eventStartDateTime = "20100728000000"`

eventEndDateTime

Syntax:	`eventEndDateTime = "`*`dateandtime`*`"`
Description:	The event end time.
Type:	String
Optionality:	Required

Allowed:	Expected format yyyymmddHHMMSS
Default:	no default
Notes:	Statistics will only be collected for the event between the start and end times.
Example:	`eventEndDateTime = "20101231000000"`

`eventWritePeriod`

Syntax:	`eventWritePeriod = <seconds>`
Description:	The period (in seconds) between successive writes of statistics data to the report file.
Type:	Integer
Optionality:	Optional (default used if not set).
Allowed:	
Default:	0
Notes:	When set to 0 the write period is synchronized to the shortest sample period defined for the statistics included in the event.
Example:	`eventWritePeriod = 30`

Event statistics

Each event can collect more than one statistic.

For example the event "EventGood" is a contest where you can vote for three contestants. The statistic for each contestant is collected separately to determine the winner.

Each event statistic accepts the following parameters:

`applicationName`

Syntax:	`applicationName = "`*`name`*`"`
Description:	The name of application to which the statistic belongs.
Type:	String
Optionality:	Required
Allowed:	
Default:	no default
Notes:	
Example:	`applicationName = "TELEVOTING"`

`statisticName`

Syntax:	`statisticName = "`*`name`*`"`
Description:	The name of the statistic.
Type:	String
Optionality:	Required
Allowed:	
Default:	no default
Notes:	
Example:	`statisticName = "Stat1"`

Statistics

As well as event statistics, smsStatisticsWriter statistics as a whole are collected, irrespective of events.

Each statistic accepts the following parameters:

`applicationName`

Syntax:	`applicationName = "`*`name`*`"`
Description:	The name of application to which the statistic belongs.
Type:	String
Optionality:	Required
Allowed:	
Default:	no default
Notes:	
Example:	`applicationName = "TELEVOTING"`

`statisticName`

Syntax:	`statisticName = "`*`name`*`"`
Description:	The name of the statistic.
Type:	String
Optionality:	Required
Allowed:	
Default:	no default
Notes:	
Example:	`statisticName = "Stat1"`

`resetStatisticOnStartup`

Syntax:	`resetStatisticOnStartup = `*`true`*`|`*`false`*
Description:	Reset statistic on start up.
Type:	Boolean
Optionality:	Optional (default used if omitted)
Allowed:	• true, or • false
Default:	true
Notes:	
Example:	`resetStatisticOnStartup = false`

`statWritePeriod`

Syntax:	`statWritePeriod = <seconds>`
Description:	The period (seconds) between successive writes of statistics data to the report file.
Type:	Integer
Optionality:	Optional (default used if not set).
Allowed:	
Default:	0
Notes:	When set to 0 the write period is synchronized to the sample period defined for the statistic.

Example:	`statWritePeriod = 0`

statResetBaseTime

Syntax:	`statResetBaseTime = "<date and time>"`
Description:	Sets a specific date and time for setting the statistic to zero.
Type:	String
Optionality:	Required
Allowed:	
Default:	None
Notes:	Date and time format is yyyymmddHHMMSS The statistic **resetInterval** parameter timing starts from this parameter's date and time.
Example:	`statResetBaseTime = "20110511130000"`

resetInterval

Syntax:	`resetInterval = {<days>,<hours>,<minutes>,<seconds>}`
Description:	The interval between periodic resets of the statistic.
Type:	Array of parameters
Optionality:	optional (default used if not set).
Allowed:	
Default:	Global resetInterval values are used.
Notes:	This interval is started from the date and time statResetBaseTime if it is present.
Example:	

```
resetInterval = {
    days = 0
    hours = 0
    minutes = 4
    seconds = 0
}
```

Event name file format

The smsStatisticsWriter process creates a file format as detailed below for an event configured in SMS.smsStatisticsWriter.Events.eventName:

================================

"*Event Name*"

"*Start Time*"

"*Stop Time*"

================================

[Automatic statistic reset *Time Stamp*]

Timestamp

[Automatic statistic reset *Time Stamp*]

"*Statistic ID*"   *Statistics Count*

[Automatic statistic reset *Time Stamp*]

"*Statistic ID*"   *Statistics Count*

….

Timestamp

[Automatic statistic reset *Time Stamp*]
"*Statistic ID*"   *Statistics Count*
[Automatic statistic reset *Time Stamp*]
"*Statistic ID*"   *Statistics Count*

For example:

```
================================
"Strictly Come Dancing"
"20:00:00"
"20:30:00"
================================
Automatic statistic reset 21-04-2010 20:00:00
21-04-2010 20:00:00
"Contestant A"   0
"Contestant B"   0
"Contestant C"   0
21-04-2010 20:01:00
"Contestant A"   300
"Contestant B"   20
"Contestant C"   50
…
21-04-2010 20:20:00
"Contestant A"   1200
Automatic statistic reset 21-04-2010 20:20:00
"Contestant B"   0
"Contestant C"   95
…
21-04-2010 20:30:00
"Contestant A"   15345
"Contestant B"   12789
"Contestant C"   120
```

Statistics file format

The smsStatisticsWriter process creates a file format as detailed below for a statistic configured in SMS.smsStatisticsWriter.Statistics:

================================

"Application ID"

"*Statistic ID*"

================================

[Automatic statistic reset *Timestamp]*

Timestamp   *Statistics Count*

….

[Automatic statistic reset *Timestamp]*

Timestamp   *Statistics Count*

….

[Automatic statistic reset *Timestamp*]

Timestamp   *Statistics Count*

For example:

```
================================
"TPSA"
"Service A"
================================
01-01-2010 00:00:00     0
01-01-2010 00:00:30     2
…
Automatic statistic reset 31-01-2010 12:00:00
31-01-2010 12:00:00      0
31-01-2010 12:00:30      3
…
31-01-2010 23:00:00     4000
```

smsStatsThreshold

Purpose

The smsStatsThreshold polls the database for updates to the SMF_STATISTICS table. It compares the values against threshold rules defined in the SMF_STATISTICS_RULE table, and raises an alarm if the threshold is exceeded. It inserts the alarm into the SMF_ALARM_MESSAGE table in the SMF database.

For more information about how to define new statistics threshold rules, see the *Service Management System User's Guide*. New threshold rules are automatically recognised by the program.

Startup

In an unclustered installation, this task is started by entry sms6 in the inittab, via the shell script:

```
/IN/service_packages/SMS/bin/smsStatsThresholdStartup.sh
```

In a clustered installation, this task is started by the cluster software, via the shell script:

```
/IN/service_packages/SMS/bin/smsStatsThresholdCluster.sh
```

Parameters

The smsStatsThreshold supports the following command-line options:

Usage:

```
smsStatsThreshold -u <usr/pwd> -s <secs>
```

The available parameters are:

`-u <usr/pwd>`

Syntax:	`-u <usr/pwd>`
Description:	The Oracle user and password pair.
Type:	String
Optionality:	Optional (default used if not set).
Allowed:	
Default:	/
Notes:	
Example:	

`-s <secs>`

Syntax:	`-s <secs>`
Description:	The number of seconds to sleep between database checks.
Type:	Integer
Optionality:	Optional (default used if not set).
Allowed:	
Default:	60
Notes:	
Example:	

Failure

Alarm messages derived from statistics values will not appear in the alarm system.

Output

The smsStatsThreshold writes error messages to the system messages file, and also writes additional output to **/IN/service_packages/SMS/tmp/smsStatsThreshold.log**.

smsSendConfig.sh

Purpose

smsSendConfig.sh resides on the source node (for example, an SMS) and performs the following functions:

- Archives current master XML file
- Stores audit information
- Sets link to current archived file
- Converts XML format to derived **eserv.config** using the script **cmnConfigXmlConvert.sh**
- Sends derived **eserv.config** file to target node using scp.

About database connections

smsSendConfig.sh connects to the database on a local or a remote SMS node by using the user credentials set in the following environment variables in **smsConfigVariables.sh**:

- SMP_DB_USER_NAME
- SMP_DB_PASSWORD
- SMP_DB_CONNECT_STRING

For connections to a:

- Local database, specify the username and password by setting the SMP_DB_USER_NAME and SMP_DB_PASSWORD variables. You can set only the user name in the SMP_DB_USER_NAME variable, if required.
- Remote database, specify the username and password by setting the SMP_DB_USER_NAME and SMP_DB_PASSWORD variables, and specify the SID of the remote database in the SMP_DB_CONNECT_STRING variable. You can set the SMP_DB_USER_NAME and the SMP_DB_CONNECT_STRING variables only, if required.
- Local or a remote database by using the Oracle wallet secure external password store, specify only the TNS connection string in the SMP_DB_CONNECT_STRING variable, where the connection string is the alias defined for the username and password credentials in the external password store. This alias can be either a TNS name or a service name from **tnsnames.ora**. The SMP_DB_CONNECT_STRING variable has the following format: "\@*connect_string*".

Note: If you do not set any of these variables, **smsSendConfig.sh** connects to the database by using the default value of "/".

Startup

smsSendConfig.sh is started by smsConfigDaemon (using the -m parameter). It is driven by the system and is not intended to be changed by the user.

Configuration

For more information on the parameters used by smsSendConfig.sh, see *smsConfigDaemonScript Configuration* (on page 104).

Failure

If smsSendConfig.sh fails, deployment process for the eserv.config on the source node will fail. Consequently, no updates will be sent to the target node. Appropriate alarm messages are generated.

Output

The **smsSendConfig.sh** and its sub-scripts write error messages to the system messages file, and also write additional output to **/IN/service_packages/SMS/tmp/smsConfigDaemonMaster.log**.

smsTaskAgent

Purpose

smsTaskAgent is a CORBA server that performs various utility functions as requested by the SMS Java screens, including:

- Create/transfer new **replication.config** file
- Change Oracle password for Convergent Charging Controller screens user
- Perform data consistency checks with remote nodes

CORBA service port

At startup, smsTaskAgent publishes its IOR string via the smsNamingServer. If not specified, the IP port number on which the CORBA service is provided will be determined by the CORBA framework. In most installations, a firewall is used to protect the SMS host, and hence the CORBA service port must be fixed. Use the `-s` parameter for this purpose.

Startup

In an unclustered installation, this task is started by entry sms8 in the inittab, via the shell script:

```
/IN/service_packages/SMS/bin/smsTaskAgentStartup.sh
```

In a clustered installation, this task is started by the cluster software, via the shell script:

```
/IN/service_packages/SMS/bin/smsTaskAgentCluster.sh
```

If there is no local SMF database and smsTaskAgent connects to the Oracle database only on a remote SMS, remove or comment out the following lines and all the lines in between:

```
"echo "`date` - Waiting for DB SMF""

"echo "`date` - DB SMF is ready""
```

smsTaskAgent configuration in eserv.config

You configure smsTaskAgent in the SMS, smsTaskAgent section of the **eserv.config** configuration file:

```
SMS = {
    smsTaskAgent = {
        defaultOracleProfile = "password_profile"
    }
}
```

defaultOracleProfile

Syntax:	`defaultOracleProfile = "`*`password_profile`*`"`
Description:	The name of the Oracle profile to allocate to new users created through the User Management screen in the SMS UI. The SMS uses the verification function for the allocated Oracle profile to check that the entered password is acceptable. The SMS also applies the verification function whenever the system administrator attempts to change a user's password through the SMS UI. If the entered password is rejected, the SMS displays an error message. You specify the error message text in the `passwordPolicyMessage` Java applet parameter. See *passwordPolicyMessage* (on page 89) for more information.
Type:	String
Optionality:	Optional (default used if not set)
Allowed:	The name of an existing Oracle profile (created by using the CREATE PROFILE command).
Default:	Use the standard Oracle profile called DEFAULT.
Notes:	For information about creating Oracle profiles and using the CREATE PROFILE command, see the Oracle Database online documentation.
Example:	`defaultOracleProfile = "password_profile"`

Command line parameters

smsTaskAgent supports the following command line options:

Usage:

```
smsTaskAgent [-c] [-i ior_host] [-p ior_port] [-u usr/pwd] [-t trans_host] [-s
trans_port] [-w secs]
```

The available parameters are:

`-c`

Syntax:	`-c`
Description:	Use secure shell and secure copy (ssh and scp).
Type:	Boolean
Optionality:	Optional (default used if not set)
Allowed:	
Default:	Use standard connection and copy
Notes:	
Example:	

`-i ior_host`

Syntax:	`-i` *ior_host*
Description:	The IOR listener host to connection to.
Type:	String
Optionality:	Optional (default used if not set)
Allowed:	
Default:	localhost
Notes:	
Example:	`-i produsms`

`-p port`

Syntax:	`-p` *port*
Description:	The port on the IOR listener host to connect to.
Type:	Integer
Optionality:	Optional (default used if not set)
Allowed:	
Default:	5556
Notes:	
Example:	`-p 13579`

`-u usr/pwd`

Syntax:	`-u` *usr/pwd|/@connect_string*
Description:	The username and password, or the connection string, to use for connections to the Oracle database on a local or a remote SMS node.
Type:	String
Optionality:	Optional (default used if not set)
Allowed:	For connections to a: • Local or remote database by using user credentials, specify the user and password, or specify '/' for passwordless connections • Local or a remote database by using the Oracle wallet secure external password store, specify only the TNS connection string where the TNS connection string is the alias defined for the username and password credentials in the external password store. This alias can be either a TNS name or a service name from **tnsnames.ora**.

Default:	/
Notes:	When smsTaskAgent invokes repConfigWrite in order to create the **replication.config** file, the specified user credentials are passed down to reConfigWrite as the `-user` argument.
Example:	`-u SMF/SMF`

`-t trans_host`

Syntax:	`-t` *trans_host*
Description:	The CORBA transport host to connect to.
Type:	String
Optionality:	Optional (default used if not set)
Allowed:	
Default:	NULL
Notes:	
Example:	

`-s trans_port`

Syntax:	`-s` *trans_port*
Description:	The port on the CORBA transport host to connect to.
Type:	Integer
Optionality:	Optional (default used if not set)
Allowed:	
Default:	0
Notes:	
Example:	

`-w secs`

Syntax:	`-w` *secs*
Description:	The number of seconds smsTaskAgent waits for a consistency check update before timing out and abandoning a consistency check.
Type:	Integer
Optionality:	Optional (default used if not set)
Allowed:	
Default:	20
Notes:	
Example:	`-w 120`

Failure

If smsTaskAgent fails, then user attempts to perform the tasks served by this smsTaskAgent will do nothing, and will display a message box indicating failure after approximately five minutes.

Output

smsTaskAgent writes error messages to the system messages file, and also writes additional output to **/IN/service_packages/SMS/tmp/smsTaskAgent.log**.

smsTrigDaemon

Purpose

smsTrigDaemon manages control plan execution requests. It runs on the SMS platform.

smsTrigDaemon accepts control plan execution requests from either a remote PI client or the Java management screens. It forwards requests to ACS through the xmlTcapInterface on the SLC platform. An indication of whether or not the requests were successful passes back from the ACS to the initiating client.

Startup

In an unclustered installation, this task is started by entry sm11 in the inittab, via the shell script:

```
/IN/service_packages/SMS/bin/smsTrigDaemonStartup.sh
```

In a clustered environment this task is started by the binary startSmsTrigDaemon which is located in:

```
/opt/ESERVSmsTrigDaemon/util/startSmsTrigDaemon
```

Note: startSmsTrigDaemon must be manually run as root in a clustered environment. smsTrigDaemon is then added as a cluster resource.

Location

This binary is located on all nodes.

Parameters

smsTrigDaemon is configured by the following parameters from the triggering section of **eserv.config**:

Usage:

```
triggering = {
    oracleLogin = "userName/password"
    useORB = true|false
    listenPort = portNumber
    slcBusyTimeout = seconds
    useFIFO = true|false
    extraFIFO = [
        "1stPath"
        "2ndPath"
            …
            …
            …
        "nthPath"
    ]
    scps = [
        "1stHostAddress:1stPortNumber"
        "2ndHostAddress:2ndPortNumber"
                …
                …
                …
        "nthHostAddress:nthPortNumber"
    ]
}
```

Available parameters are:

`oracleLogin`

Syntax:	`oracleLogin = "`*usr/pwd*`"`
Description:	The Oracle user name and password that smsTrigDaemon uses when connecting to the database.
Type:	String
Optionality:	Required
Allowed:	
Default:	/
Notes:	
Example:	

`useORB`

Syntax:	`useORB = <true|false>`
Description:	Whether smsTrigDaemon accepts incoming CORBA requests.
Type:	Boolean
Optionality:	Required
Allowed:	true — smsTrigDaemon accepts incoming CORBA requests. false — smsTrigDaemon refuses incoming CORBA requests.
Default:	false
Notes:	
Example:	

`listenPort`

Syntax:	`listenPort = `*portNumber*
Description:	The IP port on which smsTrigDaemon listens for CORBA requests.
Type:	Integer
Optionality:	Required
Allowed:	0 - 65535
Default:	0
Notes:	If set to 0, any available port is used.
Example:	

`slcBusyTimeout`

Syntax:	`slcBusyTimeout = `*seconds*
Description:	The number of seconds before connections to the SLC or VWS nodes will time out.
Type:	Integer
Optionality:	Optional (default used if not set)
Allowed:	
Default:	10
Notes:	
Example:	`slcBusyTimeout = 15`

useFIFO

Syntax:	`useFIFO = true|false`
Description:	Whether smsTrigDaemon accepts FIFO transport layer incoming requests.
Type:	Boolean
Optionality:	Required
Allowed:	true — smsTrigDaemon accepts FIFO transport layer incoming requests false — smsTrigDaemon refuses FIFO transport layer incoming requests
Default:	false
Notes:	If set to false, the *extraFIFO* (on page 145) parameter array is ignored. If set to false and *useORB* (on page 144) is also set to false, smsTrigDaemon will do nothing.
Example:	

extraFIFO

Syntax:

```
extraFIFO = [
    "dir"
    ...
]
```

Description:	The paths that smsTrigDaemon should create for extra FIFOs.
Type:	Parameter array
Optionality:	Optional
Allowed:	
Default:	Empty set
Notes:	If *useFIFO* (on page 145) is set to false, this parameter array is ignored.

Example:

```
extraFIFO = [
    "/IN/service_packages/SMS/tmp/trg-req-3005"
    "/IN/service_packages/SMS/tmp/trg-req-3006"
]
```

scps

Syntax:

```
scps = [
    "ip:port"
    ...
]
```

Description:	The Internet Protocol (IP) address and port number for each SLC to which the smsTrigDaemon connects. If you specify an IP version 6 (IPv6) address and port combination, then you must enclose the IPv6 address in square brackets [], see example for details.
Type:	Array
Optionality:	The scps parameter array is optional. In any row of the array, the **:***port* part is optional.
Allowed:	*ip* — Any IP address or symbolic host name. *port* — 0 - 65535
Default:	Empty set

Notes:	An example of an Internet protocol address is **`192.0.2.1`**. An example of an IPv6 address is **`2001:db8:`***`n`*`:`*`n`*`:`*`n`*`:`*`n`*`:`*`n`*`:`*`n`* where *`n`* is a group of 4 hexadecimal digits. The industry standard for omitting zeros is also allowed. An example of an address in symbolic name format is **`primary_smc`**.
Example:	

```
scps = [
     "198.51.100.1"
     "192.0.2.1:4000"
     "[2001:db8:0000:1050:0005:0600:300c:326b]:3004"
     "[2001:db8:0:0:0:500:300a:326f]:1234:SMF"
     "[2001:db8::c3]:1234:SMF"
     "2001:db8:1050:0:0:300a:0300:126c"
     "primary_smc"
     "secondary_smc:3006"
 ]
```

Failure

If smsTrigDaemon fails, then interaction with the BPL requests from the Java screens and the PI will fail.

Output

smsTrigDaemon writes error messages to the system messages file, and also writes additional output to **/IN/service_packages/SMS/tmp/smsTrigDaemon.log**.

Control plan execution requests

After receiving a control plan execution request the smsTrigDaemon follows a three-stage process:

Stage	Description
1	smsTrigDaemon attempts to connect to one of the SLCs in the following way. • If a connection to a SLC is established and not in use, that connection is used. smsTrigDaemon maintains a list of currently-open connections. • If a connection to a SLC is not established, smsTrigDaemon attempts to open one. SLCs are identified by the `scps` configuration parameter. smsTrigDaemon polls this list until it finds an available connection. If the connection fails, the next SLC in the list is tried.
2	smsTrigDaemon connects to the SLC using either port 3072 or another port specified by the `scps` parameter. • If a connection to a SLC is established and not in use, that connection is used. smsTrigDaemon maintains a list of currently-open connections. • If a connection to an SLC is not established, smsTrigDaemon attempts to open one. SLCs are identified by the `scps` configuration parameter. smsTrigDaemon polls this list until it finds an available connection. If the connection fails, the next SLC in the list is tried.

Stage	Description

3	smsTrigDaemon sends an XML message via an HTTP "`POST / HTTP/1.1`" request. The syntax of the message is:
	`<control-plan-exec>`
	`<control-plan><name of control plan></control-plan>`
	`<service-handle><service handle></service-handle>`
	`<cgpn><calling party></cgpn>`
	`<cdpn><called party></cdpn>`
	`<ext id="400"><host></ext>`
	`<ext id="401"><user></ext>`
	`<ext id="402"><more extensions></ext>`
	`...`
	`</control-plan-exec>`
	At least two extension parameters, id="400" and id="401", will be present. These represent the client's host and user names. Optional additional extension parameters can be included with labels id="402", id="403", etc.

Data consistency check

A race condition can occur after the **Save & Execute** button has been pressed in a Java management screen. The race condition exists between the SMS's replication system and execution on the SLCs of an smsTrigDaemon request.

To avoid this possibility, a data consistency check is carried out on the current subscriber before proceeding with the request.

Because it is not possible to know in advance which SLC will be selected by smsTrigDaemon, a data consistency check is performed on all replicated SLCs. A decision to carry on with the request is only made after the check has been completed.

The following steps describe the consistency check process and the criteria used to determine whether the execution will be allowed.

Stage	Description
1	The Java management screen sends to smsTaskAgent a request for a consistency check on the current subscriber.
2	The Java management screen waits until a check report is received from smsTaskAgent. The report is in the form of an HTML file.
3	The Java management screen extracts relevant information from the report, including: • The number of nodes checked. • The number of nodes that failed the check. • The number of nodes that replied to the check request. • The number of nodes that reported inconsistent data.
4	The consistency check: • succeeds, if: ▪ at least one node replied without error, and ▪ no node reported inconsistent data. • fails, if: ▪ no nodes replied without error, or ▪ one or more nodes reported inconsistent data.
5	If the check succeeds, the request proceeds.
6	If the request fails, steps 1 through 4 are repeated. After three fails, the request is cancelled and the user informed in an SMS message dialogue box.

Chapter 6

Background Processes on the SLC

Overview

Introduction

This chapter provides a description of the programs or executables used by the System as background processes on an SLC.

Executables are located in the `/IN/service_packages/SMS/bin` directory.

Some executables have accompanying scripts that run the executables after performing certain cleanup functions. All scripts should be located in the same directory as the executable.

Important: It is a prerequisite for managing these core service functions that the operator is familiar with the basics of Unix process scheduling and management. Specifically, the following Unix commands:

- init (and inittab)
- cron (and crontab)
- ps
- kill

In this chapter

This chapter contains the following topics.

smsApplyConfig.sh

Purpose

smsApplyConfig.sh resides on the target node, example SLC.

It performs the following functions:

- Backup of the live eserv.config currently used in production,
- Merges changes from derived file into live eserv.config using smsConfigSurgeon.
- Signals changes using smsSignalConfigChanges.sh.

If a SIGHUP is required, smsSignalConfigChanges.sh will in turn call smsSendSighup.sh (which will run with root permissions).

Startup

smsApplyConfig.sh is started by smsConfigDaemon (without the -m parameter). It is driven by the system and is not intended to be changed by the user.

Configuration

For more information on the parameters used by smsApplyConfig.sh, see *smsConfigDaemonScript Configuration* (on page 104).

Failure

If smsApplyConfig.sh fails, the deployment process for eserv.config on the target node will fail. Appropriate alarm messages will be generated.

Output

The **smsApplyConfig.sh** and its sub-scripts write error messages to the system messages file, and also write additional output to **/IN/service_packages/SMS/tmp/smsConfigDaemonClient.log**.

cmnPushFiles

Purpose

cmnPushFiles transfers files to specific directories on the SMS from SLCs and VWSs. The files transferred include:

- EDRs
- PIN logs

Note: Other Oracle applications also use their own instances of this process.

Startup

This task is started by entry scp1 in the inittab, using the shell script:

```
/IN/service_packages/SMS/bin/cmnPushFilesStartup.sh
```

Configuration

cmnPushFiles accepts the following command-line options:

Usage:

```
cmnPushFiles -d <dir> [-o <dir> [-a <days>]] [-f <dir>] [-F] [-P <pref>] [-S <sufx>]
-h <host> [-r <pref>] [-p <port>] [-s <secs>] [-R <secs>] [-M <secs>] [-C <secs>] [-
t <bits>] [-T] [-x] [-e] [-w <secs>]
```

-d <dir>

Syntax:	`-d <dir>`
Description:	The destination directory for files on remote machine.
Type:	String
Optionality:	Optional (default used if not set).
Allowed:	Path must start with '/' or the -r option must also be used. Cannot be the same as *-f <dir>* (on page 153).
Default:	.

Notes:	An example of a destination directory is the directory on a SLC where cmnPushFiles looks for the files to be sent to the SMS.
Example:	

`-P <dir>`

Syntax:	`-P <dir>`
Description:	The file prefix to match on.
Type:	String
Optionality:	
Allowed:	
Default:	
Notes:	
Example:	

`-S <sufx>`

Syntax:	`-S <sufx>`
Description:	The file suffix.
Type:	String
Optionality:	
Allowed:	
Default:	
Notes:	
Example:	

`-r <pref>`

Syntax:	`-r <pref>`
Description:	The remote directory prefix.
Type:	String
Optionality:	Optional (default used if not set).
Allowed:	
Default:	null
Notes:	Required if *-d <dir>* (on page 150) is a relative directory.
Example:	

`-h <host>`

Syntax:	`-h <host>`
Description:	The hostname of the remote machine.
Type:	String
Optionality:	Required
Allowed:	
Default:	null
Notes:	If set, a hostname must be specified.
Example:	

`-p <port>`

Syntax:	`-p <port>`
Description:	The port number on the remote machine on which cmnReceiveFiles will listen for receiving files.
Type:	Integer
Optionality:	Optional (default used if not set).
Allowed:	port Port to connect to. -1 Use stdin and stdout.
Default:	2027
Notes:	
Example:	

`-s <secs>`

Syntax:	`-s <secs>`
Description:	The number of seconds for the sleep period.
Type:	Integer
Optionality:	Optional (default used if not set).
Allowed:	
Default:	15
Notes:	
Example:	

`-t <bits>`

Syntax:	`-t <bits>`
Description:	The number of bits per second to start throttling at.
Type:	Integer
Optionality:	Optional (default used if not set).
Allowed:	
Default:	0 (no throttling)
Notes:	
Example:	

`-w <secs>`

Syntax:	`-w <secs>`
Description:	The number of seconds to wait for success.
Type:	Integer
Optionality:	Optional (default used if not set).
Allowed:	
Default:	30
Notes:	
Example:	

`-x`

Syntax:	`-x`
Description:	Whether to use hostname-prefixing on remote filenames.

Type:	Boolean
Optionality:	Optional (default used if not set).
Allowed:	set (false) — Don't use prefixing. not set (true) — Use prefixing.
Default:	true
Notes:	
Example:	

`-o <dir>`

Syntax:	`-o <dir>`
Description:	The directory to transfer sent files to.
Type:	String
Optionality:	Optional (default used if not set).
Allowed:	directory — The directory to store transferred files in. null — Delete the transferred files, do not store them.
Default:	null (file deleted)
Notes:	
Example:	

`-f <dir>`

Syntax:	`-f <dir>`
Description:	The retry directory.
Type:	String
Optionality:	Optional (default used if not set).
Allowed:	Cannot be the same as *-d <dir>* (on page 150).
Default:	null (no retry directory)
Notes:	
Example:	

`-F`

Syntax:	`paraMeter = <>`
Description:	Use fuser to not move files in use.
Type:	Boolean
Optionality:	Optional (default used if not set).
Allowed:	set (true) — Use fuser. not set (false) — Don't use fuser.
Default:	false
Notes:	
Example:	

`-a <days>`

Syntax:	`-a <days>`
Description:	The number of days old a transferred file can be before it is deleted.

Type:	Integer
Optionality:	Optional (default used if not set).
Allowed:	positive integer
	-1 never delete files.
Default:	-1
Notes:	This parameter only relevant when *-o <dir>* (on page 153) option is specified.
Example:	

`-e`

Syntax:	`-e`
Description:	Which mode to run in.
Type:	Boolean
Optionality:	Optional (default used if not set).
Allowed:	set (false) Run in non-daemon mode. Execute file transfer only once, then exit.
	not set (true) Run in daemon mode.
Default:	not set
Notes:	
Example:	

`-R <secs>`

Syntax:	`-R <secs>`
Description:	The number of seconds before Initial retry period starts.
Type:	Integer
Optionality:	Optional (default used if not set).
Allowed:	
Default:	15
Notes:	
Example:	

`-M <secs>`

Syntax:	`-M <secs>`
Description:	The maximum number of seconds for the retry period to continue.
Type:	Integer
Optionality:	Optional (default used if not set).
Allowed:	
Default:	900
Notes:	
Example:	

`-C <secs>`

Syntax:	`-C <secs>`
Description:	The number of seconds for the cleanup period.
Type:	Integer

Optionality:	Optional (default used if not set).
Allowed:	
Default:	1800
Notes:	
Example:	

`-T`

Syntax:	`-T`	
Description:	Whether or not to move recursively.	
Type:	Boolean	
Optionality:	Optional (default used if not set).	
Allowed:	set	Tree move: recursive into subdirectories.
	not set	
Default:	true	
Notes:		
Example:		

Example

This text shows an example of the command line options for cmnPushFiles.

```
cmnPushFiles -d /IN/service_packages/SMS/cdr/closed -f
/IN/service_packages/SMS/cdr/retry -r /IN/service_packages/SMS/cdr/received -h
prodsmp1.telcoexample.com -s 10 -p 2028 -S cdr -w 20
```

Failure

If cmnPushFiles fails, EDRs will accumulate in:

/IN/service_packages/SMS/cdr/current/

cmnPushFiles will send error messages to the syslog and the cmnPushFiles log.

Output

The cmnPushFiles writes error messages to the system messages file, and also writes additional output to this default location:

/IN/service_packages/SMS/tmp/cmnPushFiles.log

infMaster

Purpose

The infMaster provides resilience for replication in case the smsMaster fails. For more information, see Inferior Master.

The infMaster is only used in the unclustered configuration.

Note: The infMaster does not replicate Alarms or Statistics.

Startup

This task is started by entry scp2 in the inittab, via the shell script:

```
/IN/service_packages/SMS/bin/infMasterStartup.sh
```

Parameters

The infMaster supports the following command-line options:

Usage:

```
infMaster [-maxpending <number>]
```

The available parameters are:

Parameter	Default	Description
-maxpending <number>	10000	This sets the maximum number of pending updates which will be queued in the infMaster's memory.

Failure

If the infMaster fails, no functionality will be affected unless the infMaster would normally be required to operate as the Superior Master (that is, the smsMaster and all other infMasters with higher node numbers were unavailable). In this case, replication will not work.

The infMaster will send error messages to syslog and infMaster.log.

Output

The infMaster writes error messages to the system messages file, and also writes additional output to **/IN/service_packages/SMS/tmp/infMaster.log**.

smsAlarmDaemon

Purpose

The smsAlarmDaemon runs on all alarm-managed nodes in the SMS system, including the SMS nodes. The role of the smsAlarmDaemon is to gather alarms from the following sources:

- System error log (/var/adm/syslog.log or /var/log/syslog)
- Oracle Standard DB error log
- Sigtran SUA logs (/IN/service_packages/SLEE/stats) [If installed]

On the SMSs, the resultant error messages are written directly into the SMF_ALARM_MESSAGE table in the SMF. When run on other nodes, replication is used to update the SMF_ALARM_MESSAGE table.

Startup

This task is started by entry scp4 in the inittab, via the shell script:

```
/IN/service_packages/SMS/bin/smsAlarmDaemonScpStartup.sh
```

Configuration

smsAlarmDaemon accepts the following command-line arguments.

Usage:

```
smsAlarmDaemon [-l seconds] [-h seconds] [-n number] [-m number] [-p] [-d] [-a path]
[-r node] [-u user/pass] [-f] [-i] [-g] [-c number] [-t seconds]
```

The available parameters are:

Parameter	Default	Description
`-a` *path*	Null	Propagate alarms from the specified Oracle alert log to the database. By default, smsAlarmDaemon does not propagate alarms from the Oracle alert log.
`-c` *number*	1	Commit Rate. The number of inserts before committing to the database.
`-d`	Sort messages	Disable sorting of messages in the buffer by severity. Specifically, messages are kept in the buffer and subsequently written into the SMF database, in the same sequence in which they are received.
`-f`	No filtering	Filtering. Delete duplicate alarms and increase the alarm count.
`-g`	Uses local time	GMT timezone. Use GMT instead of local time.
`-h` *seconds*	60	Heartbeat message. Will be forced to be greater or equal to time period (seconds).
`-i`	Use fuzzy matching	Filtering type. Use exact matching (rather than fuzzy matching). Indicates that duplicate matches should be performed on text only (that is, excluding digits). **Note:** Only valid when used in conjunction with **-f**.
`-l` *seconds*	2	Filter Period. Duration between linked-list checks (in seconds).
`-n` *number*	5	Filter Number. The number of alarm messages allowed within the time period. **Allowed values:** Integers
`-m` *number*	1000	Maximum number of alarm messages to buffer. **Allowed values:** Integers 1-1000000
`-p`	Do not drop messages	Drop low-priority messages when the buffer is full. Specifically, when **-m** *number* messages have been received but it is not yet time to write the buffer contents to the SMS database, low priority messages in the buffer are dropped in favor of higher-priority messages that may be received on its input stream.
`-r` *node*	Direct to the Oracle DB	Replication node. Specify the replication requester node.
`-t` *seconds*	1	Commit interval. The maximum interval between database commits (in seconds).
`-u` *user/pass*	/	Use the supplied Oracle user/password pair.

Usage example

Here is an example of using smsAlarmDaemon:

```
smsAlarmDaemon -l 5 -h 30 -n 10 -m 2000 -p -d -a /volB/home/saich -r 750 -u
smf/smf -f -i -g -c 2 -t 2
```

- Filter Period (-l) = 5 seconds
- Heart beat (-h) = Yes every 30 seconds
- Filter Number (-n) = 10 each period
- Max number(-m) = 2000 records
- Drop low priority messages (-p) = true
- Sort messages by severity (-d) = false
- Oracle Alert Log location (-a) = /volB/home/saich
- Rep node (-r) = 750
- Oracle User (-u) = smf/smf
- Filtering (-f) = Multiple alarms combined
- Filtering type (-i) = Exact match
- GMT timezone (-g) = Yes
- Commit Rate (-c) = every 2 number of inserts
- Commit Interval (-t) = every 2 seconds if 2 records not reached

Failure

The smsAlarmDaemon on each alarm-managed node in the installation will by default generate a health-check alarm once per minute. These health check alarms will be relayed in the same fashion as all other alarms.

If these health check alarms are not received at the target destination, then the smsAlarmDaemon may have failed, and should be investigated.

Output

The smsAlarmDaemon writes error messages to the system messages file, and also writes additional output to **/IN/service_packages/SMS/tmp/smsAlarmDaemonScp.log**.

smsLogCleaner

Purpose

smsLogCleaner archives the following types of log files:

- Convergent Charging Controller process log files (/IN/service_packages/*Product*/tmp/*Process*.log)
- System log files (syslog)

For more information, see *System Administrator's Guide*.

Startup

This task is run in the crontab for smf_oper. By default, it runs at 30 minutes past each hour. It is run via the shell script:

```
/IN/service_packages/SMS/bin/smsLogCleanerStartup.sh
```

Parameters

smsLogCleaner supports the following command-line options:

Usage:

```
smsLogCleaner -c configuration_file -d days -s storage_file [-h]
```

The available parameters are:

Parameter	Default	Description
-c *configuration_ file*	**logjob.conf**	The name of the configuration file to use.
-d days	7	How often to clean the archive (in days).
-s *storage_file*	**storage.txt**	The name of the storage file to use.
-h		Provides help information.

At installation, the cronjob is configured to execute by default with the following command line parameters:

`-c` **/IN/service_packages/SMS/etc/logjob.conf**

`-s` **/IN/service_packages/SMS/tmp/sms_storage.txt**

`-d 7`

An operator can change these values, subject to disk storage availability and site-specific archiving policies.

Failure

If the process is not running, log files in the following directory will accumulate in size and age beyond the expected values.

```
/IN/service_packages/SMS/tmp
```

Output

The smsLogCleaner run by smf_oper writes error messages to the system messages file, and also writes additional output to **/IN/service_packages/SMS/tmp/smsLogCleaner.log**.

logjob.conf

The logjob.conf configuration file has the following format:

```
log <file> age <hrs> size <size> arcdir <dir> logonce
```

The available parameters are:

Parameter	Description
log <file>	The full directory path and name of the file to be cleaned. You can include the '*' wildcard in the file name if required. **Example:** `log /IN/service_packages/SMS/tmp/smsNamingServer.log` **Tip:** Most processes and tools document where their output is written to in their Output topic.
age <hrs>	Sets the minimum age in hours for the log file before it will be cleaned. You must set either this parameter or the size parameter. If both parameters are set, then the log file is cleaned if either condition is met. **Example:** `age 100`
size <size>	Sets the minimum size for the logfile before it will be cleaned. You must set either this parameter or the age parameter. If both parameters are set, then the log file is cleaned if either condition is met. **Examples:** `size 60K`, or `size 60M`

Parameter	Description
arcdir <dir>	The directory to use to store the old log file. If this parameter is not specified, then the log file is deleted. **Example:** `arcdir /IN/service_packages/SMS/tmp/archive`
logonce	Only specify this parameter if you just want to keep one archived version of the log file.

smsStatsDaemon

Purpose

The smsStatsDaemon program is the key component in the statistics process. The statistics process gathers and updates all statistics values through a single consistent mechanism over the network.

The smsStatsDaemon can optionally dynamically load extension libraries at runtime to provide extra functionality. This functionality includes node uptime, process uptime, and database row counts.

Startup

This task is started by entry scp3 in the inittab, via the shell script:

```
/IN/service_packages/SMS/bin/smsStatsDaemonStartup.sh
```

smsStatsDaemon configuration

The stats daemon can run in one of two modes:

- standard mode or
- legacy mode.

The standard mode uses command line options to configure the smsStatsDaemon, and uses the SMF_STATISTICS_DEFN table in the SMF database to define the statistics which should be collected.

The legacy mode is configured using a combination of command line parameters and a configuration file. The configuration file defines the statistics which should be collected.

Depending on which mode the smsStatsDaemon is running in, a different set of parameters will be used.

Parameters

The command line parameters for the smsStatsDaemon are:

Usage:

```
smsStatsDaemon [-e <secs>] [-u <usr/pwd>] [-f <dir/file>] [-v] [-r <node>] [-d
<size>] [-h <ratio>] [-w] [-F] [-m <size>] [-i] [-S] [-T] [-C <n>]
```

Or for legacy form:

```
smsStatsDaemon [-c <dir>] [-a <dir>] [-t <secs>] [-s <Kb>] [-e <secs>] [-f
<dir/file>] [-v] [-d <size>] [-h <ratio>] [-w] [-F] [-m <size>] [-i] [-S] [-T] [-C
<n>]
```

The available parameters are:

`-e secs`

Syntax: `-e` *`secs`*

Description: The minimum number of seconds between logging statistic counts of zero.

Type:	Integer
Optionality:	Optional (default used if not set).
Allowed:	
Default:	0
Notes:	This only applies to collection mode 1 (always report).
Example:	

`-f dir/file`

Syntax:	`-f` *`dir/file`*
Description:	The stats configuration file (full path, includes file name).
Type:	String
Optionality:	Optional (stats file not used if not set).
Allowed:	
Default:	null
Notes:	
Example:	

`-v`

Syntax:	`-v`
Description:	Use verbose mode (provide more information while processing).
Type:	Boolean
Optionality:	Optional (default used if not set).
Allowed:	
Default:	Do not use verbose mode.
Notes:	Usually used for debuggin set-up problems.
Example:	

`-d rows`

Syntax:	`-d` *`rows`*
Description:	The number of rows for dynamic stats table.
Type:	Integer
Optionality:	Optional (default used if not set).
Allowed:	> 1
Default:	500
Notes:	The dynamic stats table is the shared memory hash table used to contain the statistics information. The size of this table varies with the number of statistics being collected.
Example:	

`-h ratio`

Syntax:	`-h` *`ratio`*
Description:	The ratio of the size of the hash index to the dynamic stats table.
Type:	Integer
Optionality:	Optional (default used if not set).
Allowed:	

Default:	2
Notes:	The dynamic stats table is the shared memory hash table used to contain the statistics information. The size of this table varies with the number of statistics being collected.
Example:	

`-F`

Syntax:	`-F`
Description:	Only do SMS-specific statistic collection.
Type:	Boolean
Optionality:	Optional (default used if not set).
Allowed:	
Default:	Collect all statistics.
Notes:	Only set when smsStatsDaemon is running on an SMS. The SMF_STATISTICS_EXTN table specifies for each extension statistic whether it is an SMS stat or not. Also the shared memory hash table specifies whether each statistic is an SMS stat. Hence the -F option can do only SMS stats or all stats. **Examples:** Uptime of *smsMaster* (on page 120) is an SMS stat, uptime of *updateLoader* (on page 168) is not an SMS stat.
Example:	

`-m size`

Syntax:	`-m` *`size`*
Description:	The size of the details column in the dynamic stats table.
Type:	Integer
Optionality:	Optional (default used if not set).
Allowed:	
Default:	80
Notes:	The dynamic stats table is the shared memory hash table used to contain the statistics information.
Example:	

`-i`

Syntax:	`-i`
Description:	Silently ignore mount points that are longer than the *-m size* (on page 162) limit.
Type:	Boolean
Optionality:	Optional (not used if not set).
Allowed:	
Default:	
Notes:	
Example:	

`-S`

Syntax:	`-S`
Description:	Silently drop statistics where the details field is longer than *-m size* (on page 162).
Type:	Boolean

Optionality:	Optional (not used if not set).
Allowed:	
Default:	
Notes:	
Example:	

`-T`

Syntax:	`-T`
Description:	Truncate the detail field for statistics that exceed *-m size* (on page 162).
Type:	Boolean
Optionality:	Optional (not used if not set).
Allowed:	
Default:	
Notes:	
Example:	

`-C n`

MERSyntax:	`-C` *`n`*
Description:	Sets the global configuration variable mergeCpuStats to the value of "n". This defines whether individual CPU statistics will be output or whether CPU statistics will be summed. For summed statistics, the number of CPUs is also output.
Type:	Integer
Optionality:	Optional (default used if not set).

Allowed:	n = 0 output individual CPU statistics
	n = 1 sum CPU statistics
	Any other value of n will be ignored, and default behavior will be used

Default:	0 - output individual CPU statistics
Notes:	If MERGECPUSTATS config file entry is set, it overrides the default behaviour. If the command line switch (-C) is set, then it will override the MERGECPUSTATS config file entry.
Example:	`-C 1`

Parameters for standard mode

This parameters are used with the general parameters when smsStatsDaemon is running in standard mode.

Note: These parameters cannot be used with the *Parameters for legacy mode* (on page 164).

`-u usr/pwd`

Syntax:	`-u` *`usr/pwd`*
Description:	The userid and password to use to log into the SMF database.
Type:	String
Optionality:	Optional (default used if not set).
Allowed:	
Default:	/
Notes:	

Example:

`-r node`

Syntax:	`-r` *`node`*
Description:	The replication node number smsStatsDaemon should use.
Type:	Integer
Optionality:	Optional (default used if not set).
Allowed:	
Default:	-1
Notes:	
Example:	

`-w`

Syntax:	`-w`
Description:	Do Row Count statistic collection.
Type:	Boolean
Optionality:	Optional (not used if not set).
Allowed:	
Default:	
Notes:	Provides statistics of the number of rows in selected database tables as defined in the SMF_STATISTICS_EXTN table. This type of statistic should only be collected on the primary SMS node.
Example:	

Parameters for legacy mode

These parameters can be used if smsStatsDaemon is being used in legacy mode.

Note: These parameters cannot be used with the *Parameters for standard mode* (on page 163).

`-c dir`

Syntax:	`-c` *`dir`*
Description:	Current statistics file directory.
Type:	String
Optionality:	Optional (default used if not set).
Allowed:	
Default:	/tmp
Notes:	
Example:	

`-a dir`

Syntax:	`-d` *`dir`*
Description:	Archived statistics file directory.
Type:	String
Optionality:	Optional (default used if not set).
Allowed:	
Default:	/tmp

Notes:
Example:

`-t secs`

Syntax:	`-t` *`secs`*
Description:	The maximum number of seconds a statistics file can be open.
Type:	Integer
Optionality:	Optional (default used if not set).
Allowed:	
Default:	1800
Notes:	
Example:	

`-s Kb`

Syntax:	`-s` *`Kb`*
Description:	The maximum number of Kb a stats file can reach.
Type:	Integer
Optionality:	Optional (default used if not set).
Allowed:	
Default:	10
Notes:	
Example:	

Failure

If the smsStatsDaemon fails, statistics on that SLC will not be processed. When the smsStatsDaemon is restarted the statistics will be processed.

Output

The smsStatsDaemon writes error messages to the system messages file, and also writes additional output to **/IN/service_packages/SMS/tmp/smsStatsDaemon.log**.

Measurement IDs - standard mode

Measurement IDs for the statistics which should be collected are loaded from the SMF_STATISTICS_DEFN and SMF_STATISTICS_EXTN tables in the Oracle Standard DB instance given by ORACLE_SID. Setting the TWO_TASK variable allows a machine without a database instance running access a database on a remote machine. One application of this may be to allow monitoring of a remote disaster recovery machine.

Statistics shared memory

The shared memory area contains an index to the statistics measurements it contains. Each measurement has an accumulator for up to 16 SLPI instances. The single SLPI process is the only process to write to that buffer. The per-SLPI statistics counters are never reset, the smsStatsDaemon treats them as read-only.

smsStatsDaemon parameters

This table gives the type and valid values of the parameters.

Name	Type	Description
CURRENTDIR	256 characters	Where active statistics files are stored.
ARCHIVEDIR	256 characters	Where archived statistics files are stored.
OPENTIME	unsigned long	The maximum length of time a statistics file should remain open. The range of values is 1-1440 minutes.
MAXSIZE	unsigned long	The maximum size (in Kbytes) a statistics file is allowed to reach. **Note:** This is only checked after a recording period, so a file may be larger than this size.
NOTIFY	256 characters	Space separated e-mail accounts to notify when the statistics file is rotated. This value is optional. If it does not exist, no e-mail is sent.
MID	Measurement description record	Specifies a measurement to be made available.
MERGECPUSTATS	1 character	If this config file entry is set, it overrides the default behaviour. The command line switch (-C) when set, overrides both this config file entry and the default entry.

Legacy mode configuration - config file

To provide full backwards support for sites using the SMS version 1 style configuration, the use of a configuration file is optional. A configuration file will be searched for according to the following rules:

- If the -f <*config_file_loc*> parameter is specified, the config file is used. An error occurs if the specified file does not exist.
- If the -f parameter is omitted, then a search is made for a file "etc/smsStatsDaemon.cfg" or "../etc/smsStatsDaemon.cfg". If one of these files exists, then the file is used. Otherwise the smsStatsDaemon will start with the default configuration as described above.

The configuration file provides:

- Parameters (for example, max open file size, archive file directory), and
- Measurement IDs (specified using "MID=…" entries).

Note: Any configuration specified in the command line will override the details in the configuration file. The database configuration is NOT used.

Syntax for the stats_config file

For legacy sites, the syntax of the stats_config file is given.

This is an example of a stats_config file:

```
# Log file locations (trailing / is optional)
CURRENTDIR=/IN/service_packages/SMS/statistics/current
ARCHIVEDIR=/IN/service_packages/SMS/statistics/archive
# Max open time is 10 min (1-1440)
OPENTIME=10
# Max file size is 128 kb (unlimited)
```

```
MAXSIZE=128
MID=npNotFound,NP,Portability request with no target,3600,Category NF
MID=npTimeOut,NP,Exceeded regulated time-out on connect,300,No comment
MID=vpnManagement,VPN,Calls to management hotline,3600,Non-charged
MID=vpnSchedule,VPN,Calls activating scheduled routing,3600,No comment
```

Where:

#	indicates a comment. The comment character needs to start at the beginning of a line. The entire line is then ignored (up to 255 characters).
CURRENTDIR	indicates the working directory of the daemon. This is where a temporary statistics file is stored until the file size of open time exceeds the configured value.
ARCHIVEDIR	the location a closed statistics file is moved to.
OPENTIME	indicates how long a statistics file can stay active (that is, how long can the daemon keep on writing statistics into a file) in minutes.
MAXSIZE	The maximum allowable size of a statistics file in Kilobytes. **Note:** If a statistics file becomes overloaded half way through a dump, the entire record will still be written.
MID	The measurements to retrieve. Refer to the subsection on Measurements

Measurements

Measurements are specified in the configuration file using one MID command for each measurement to be defined.

MID commands are comma separated value names, that must be in the order below:

- ID
- APPLICATION
- DESCRIPTION
- PERIOD
- COMMENT
- EXTN
- KEYWORD
- DETAIL
- LIB_NAME
- FUNCTION_NAME

Examples:

An example MID line without the extension fields might be:

```
MID=vpnManagement,VPN,Calls to management hotline,3600,Non-charged
```

An example MID line with the extension fields would be:

```
MID=STATSDAEMON,SCP_SYSTEM,Uptime for smsStatsDaemon process,60,smsStatsDaemon
process uptime in
minutes,EXTN,PROCESS_UPTIME,smsStatsDaemon,libsmsextrastats.so,getProcessUptime
```

This table describes the measurement parameters.

Name	Type	Description
APPLICATION	20 characters	The application ID. This may be up to 20 characters for clarity, however the first three characters must be unique.

Name	Type	Description
COMMENT	256 characters	Textual comment relating to the statistic.
DESCRIPTION	256 characters	The textual description of the measurement.
DETAIL	80 characters	Extra data required to measure stat, i.e. process name for process uptime stats. Can be NULL (empty string).
EXTN	5 characters	The keyword 'EXTN' indicating that this mid line has the extra fields.
FUNCTION_NAME	50 characters	Function within the library (specified at LIB_NAME) to call in order to measure stat.e.g. getNodeUptime.
ID	20 characters	The measurement ID.
KEYWORD	20 characters	May be required by measurement function. e.g. UPTIME_NODE.
LIB_NAME	30 characters	Dynamic library to load to get stat measurement function. e.g. libsmsextrastats.so.
PERIOD	unsigned long	The time in seconds between each recording in the output file of this statistic. Value range is 10-31536000 (1yr).

After a change is made to the Measurement IDs, the smsStatsDaemonRep process needs to be notified via a SIGHUP. This can be performed manually, or via the smsStatsDaemonRepReload.sh script provided as part of the installation.

Updating smsStatsDaemon measurements

After a change is made to the Measurement IDs, either via the database, or via modifying "MID=…" entries in the stats_config file, the smsStatsDaemon process needs to be notified via a SIGHUP. This can be performed manually, or via the smsStatsDaemonReload.sh script provided as part of the installation.

updateLoader

Purpose

The updateLoader accepts updates from the smsMaster and makes the requested changes to the database it is configured to update. More than one updateLoader may run on each SLC. For more information about updateLoaders and replication, see *What is Replication?* (on page 19)

If the SCP data becomes out of sync with the data in the SMF, a resync can be done to ensure the SCP has the correct information. It should not be necessary to do a manual resync. The system does automatic resyncs as necessary.

There are three cases where the system will resync:

Case	Description
A Node is Out of History	Where a node is isolated SMS will hold a queue of updates for the node. In the replication.def file there is a max pending variable that gives the maximum number of updates that will be held in this queue for each SLC. If this limit is exceeded (the node is out of history) SMS will drop all the entries and force a resync of the node when it comes back on line.
New Node Added	When a new node is added to the system a replication config file will be sent to the node. This forces a resync.
Replication Changed	If the replication config file for a node is changed then a resync will be forced.

Startup

This task is started by entry scp5 in the inittab, via the shell script:

```
/IN/service_packages/SMS/bin/updateLoaderStartup.sh
```

Parameters

The updateLoader supports the following command-line options:

Usage:

```
updateLoader [-nodeid node_number] [-resync]
```

The available parameters are:

Parameter	Default	Description
`-nodeid` *`node_number`*	274	The node number of the updateLoader requesting the resync.
`-resync`		Causes the updateLoader to re-synchronise with the smsMaster.

Failure

If the updateLoader is not working, updates from the SMS to the SCP database will be unsuccessful. The SLC will continue to run on the last configuration successfully loaded from the SMS.

An error message will be logged to the syslog and the updateLoader log, and may be logged to the smsMaster log.

Output

The updateLoader writes error messages to the system messages file.

Chapter 7

Tools and Utilities

Overview

Introduction

This chapter provides a description of the operational programs or executables used by the system.

Executables are located in the `/IN/service_packages/SMS/bin` directory.

Some executables have accompanying scripts that run the executables after performing certain cleanup functions. All scripts should be located in the same directory as the executable.

In this chapter

This chapter contains the following topics.

cmnConfigSyntaxCheck

Purpose

cmnConfigSyntaxCheck is used to check that the syntax of the **eserv.config** file is correct.

Configuration

cmnConfigSyntaxCheck accepts the following command line options.

Usage:

```
cmnConfigSyntaxCheck [-v -d] filename [filename [...]]
```

The available parameters are:

Parameter	Default	Description
-v	-	Verbose mode. Displays in detail all information that is available.
-d	-	Reads and dumps the named files

Output

cmnConfigSyntaxCheck displays the results of the syntax check on the terminal.

Example: This text shows an example of a report from the cmnConfigSyntaxCheck.

```
$  cmnConfigSyntaxCheck -v filename /ACS/etc/acs.conf
cmn::FileNotFoundException: Opening config file filename
Config file syntax error: ACS/etc/acs.conf:30: Syntax Error
```

cmnSU

Purpose

cmnSU replaces the built in Solaris 11 “su” command for Convergent Charging Controller processes run from initab on Solaris 11 environments only. Run as root, it will provide a login shell to the specified user.

Configuration

cmnSU accepts the following command line options.

Usage:

```
 /cmnSU - username [arg...]
```

The available parameters are:

Parameter	Default	Description
–		Provide a login shell. Required.
username		The user to become. Required.
[*arg...*]:		The rest of the arguments to the shell

compareNode

Purpose

This command can be used to initiate a full database comparison of an SCP database with the definitive copy in the SMF database.

This is used to ensure that an SCP database has all its data consistent with the SMF database. Under normal conditions, this should always be the case, but there may be a time (for example, after multiple failures) where the System Administrator wants to check that an SLC database is consistent.

The compareNode tool requests a comparison between the contents of the SMF database and one other node, by invoking comparisonServer. This is a more time-efficient method than a resync. All the entries of all the tables that are defined to be replicated to the specified updateLoader will be compared.

A full report of the comparison is written in the report directory (REPORT DIR) on the SLC machine.

Configuration

compareNode accepts the following command line options.

Usage:

```
compareNode [-hostname hostname|-master node_num] [-with node_num] [-timeout
seconds]
```

The available parameters are:

Parameter	Default	Description
`-hostname`		Sets the hostname of the superior node in the comparison. (Optional. If used, -master must = 0, that is, if -master must be set to off.)
-master	1	Node number of the superior node in the comparison. (Optional, as the default will be used if it is not set. Or it can be turned off by setting to 0, and a hostname specified instead.) **Note:** This can be an infMaster.
`-with`	256	Node number of the updateLoader in the comparison. (Optional, as the default will be used if it is not set.)
`-timeout`	10	Number of seconds before the connection between the nodes in the comparison is timed out. (Optional, as the default will be used if it is not set.)

Example: This text shows the common usage of compareNode being run on the superior master in a node comparison.

```
compareNode -with 301
```

Failure

If compareNode fails, it will send error messages to stdout and syslog.

Output

The compareNode writes error messages to the system messages file, and also writes additional output to **/IN/html/SMS/output/***node_number/timestamp***.html**.

comparisonServer

Purpose

comparisonServer is a shell script which starts node comparisons between data in two different nodes. It is started by an SMS in response to a database comparison request.

Configuration

comparisonServer accepts the following command line options.

Usage:

```
comparisonServer node_to_compare address port
```

The available parameters are:

Parameter	Default	Description
node_to_compare		
address		
port		

Output

comparisonServer writes error messages to the system messages file, and also writes additional output to **IN/service_packages/SMS/tmp/comparisonServer.log**.

inetCompareServer

Purpose

inetCompareServer is a shell script which is run by the Replication Check screens. It uses the report configuration information from the Replication Check screen to start a node comparison (which is performed by smsCompareResyncServer). It should not be necessary to run this script by hand.

Output

inetCompareServer writes to syslog and also logs additional information (including raw Replication Check report data) to **/IN/service_packages/SMS/tmp/inetCompareServer.log**.

Example: This text shows an example of a report from the inetCompareServer.

```
Copyright (c) 2002, Oracle. Contact Oracle at support@oracle.co.nz

Input delivered through standard input. Please consult the SMS administration
guide for more information on this software. For command line options, pass '-h'
as the only option to this program.


Awaiting server control information...

Input accepted. Now running server.
Mar  4 05:01:49 smsCompareResyncServer(28781) NOTICE: Beginning comparison for node
301.
COM: Fri Mar  4 05:01:50 2005: Node 301, started processing 186 SMS and 186 SCP
records.
COM: Fri Mar  4 05:01:50 2005: Node 301, table ACS_CALL_PLAN, started processing 186
SMS and 186 SCP records.
COM: Fri Mar  4 05:01:51 2005: Node 301, table ACS_CALL_PLAN, group ACS_CALL_PLAN,
started processing 186 SMS and 186 SCP records.
COM: Fri Mar  4 05:01:52 2005: Node 301, table ACS_CALL_PLAN, group ACS_CALL_PLAN,
finished processing 186 of 186 SMS and 186 of 186 SCP records, 0 discrepancy found
in group.
COM: Fri Mar  4 05:01:52 2005: Node 301, table ACS_CALL_PLAN, finished processing
186 of 186 SMS and 186 of 186 SCP records, 0 discrepancy found in table.
COM: Fri Mar  4 05:01:52 2005: Node 301, finished processing 186 of 186 SMS and 186
SCP of 186 records, 0 discrepancy found in node.
Mar  4 05:01:54 smsCompareResyncServer(28781) NOTICE: Ending comparison for node
301.
Mar  4 05:01:54 smsCompareResyncServer(28781) NOTICE: Comparison was successful for
node 301.
Started writing index HTML file for reports.
```

```
Finished writing index HTML file for reports.
```

infoDisplayer

Purpose

infoDisplayer is an executable which can be used to display update request information results.

Configuration

infoDisplayer supports the following command-line options:

Usage:

```
infoDisplayer [-host value] [-master value] [-nodeid value] [-timeout value]
```

The available parameters are:

Parameter	Default	Description
`host`	localhost	The local host name
`master`	0	The node number of the smsMaster
`nodeID`	1	The ID of the node for which the information is to be displayed.
`timeout`	10	Period after which infoDisplayer will timeout

Output

Examples:

```
bash-2.05$ ./infoDisplayer -nodeid 999 -master 1
initialiseNode: Reading '/IN/service_packages/SMS/etc/replication.def'
initialiseNode: heartbeatPeriod 20
initialiseNode: heartbeatTimeout 20
initialiseNode: connectionTimeout 2
initialiseNode: masterPortNum 12343
initialiseNode: queueWarnThresh 5
initialiseNode: queueErrThresh 100000
initialiseNode: queueCritThresh 1000000
initialiseNode: hBTolerance 10.0
initialiseNode: commitIdleTime 0.100000
initialiseNode: commitBusyTime 10.0
initialiseNode: tcpAbortSecs 20
initialiseNode: oracleUserPass '/'
initialiseNode: reportDir '/IN/service_packages/SMS/tmp/'
initialiseNode: statusFile '/IN/html/status.html'
initialiseNode: configFilePath '/IN/service_packages/SMS/etc/replication.config'
initialiseNode: configFileName 'replication.config'
initialiseNode: node number 999
initialiseNode: node type 5
initialiseNode: s side updates 1
Nov 22 22:05:17 infoDisplayer(6589) NOTICE: Master Controller `./infoDisplayer'
process started (node 999)
```

inputBootstrap

Purpose

The purpose of the inputbootstrap binary is to produce a configuration file to be passed to the smsCompareResyncServer from **replication.config**. It is started by the comparisonServer or the resyncServer which initiates requests. It can also be executed manually from the command line.

It must be noted that this binary cannot be run with DEBUG when used with the comparisonServer. (Applicable to production environment).

Note: This binary is not intended to be run by the user. Please contact your Oracle support before attempting to do so.

Configuration

inputBootstrap accepts the following command line options.

Usage:

```
inputBootstrap -n node_id [-c config_filename] [-a ip_address] [--hex-address
ip_address] [-p port] [-r] [-u usr/pwd] [-e] [-i interval] [-h] [-b]
```

Or long form:

```
inputBootstrap --node-id=node_id [--config-file=config_filename] [--
address=ip_address] [--hex-address ip_address] [--port=port] [--preserve-ranges] [--
oracle-user=usr/pwd] [--enhanced-recovery] [--sync-marker-retry-interval=interval [-
-help] [--build-info]
```

The available parameters are:

Parameter	Default	Description
`-n` `--node-id`	none	The replication node ID for which the configuration file is produced. (Required.) **Allowed values:** integers, (any signed number of reasonable value, usually in decimal/octal/hex).
`-c` `--config-file`	/IN/service_packages/SMS/etc/replication.config	Name of the replication configuration file to use. (Optional.) **Allowed values:** string
`-a` `--address`	from file specified in config-file	The IP address of the node. This cannot be specified if the '--hex-address' option is specified. (Optional.) If specified, this will override all addresses specified in the configuration file. **Allowed values:** string
`--hex-address`	from file specified in config-file	The IP address of the node as a hex string. This cannot be specified if the '-a' ('--address') option is specified. (Optional.) If specified, this will override all addresses specified in the configuration file. **Allowed values:** string
`-p` `--port`	none	The port number to connect to at the given node. (Optional.) **Note:** This can only be used when the address is also specified. **Allowed values:** integers, (any signed number of reasonable value, usually in decimal/octal/hex).

Parameter	Default	Description

`-r`
 `--preserve-ranges`
	false when missing	Leave the data from the configuration file as it is and do not correct the group ranges. This option is implied by the -e option.
		 Allowed values:
		 • 1, on, yes, true
		 • 0, off, no, false

`-u`
 `--oracle-user`
	smf/smf	The Oracle database connection string (userid/password). (Optional.)
		 Allowed values: string

`-e`
 `--enhanced-recovery`
	false when missing	Restrict range of rows to resync. If enhanced recovery mode is possible, the smsMaster process sets this option.
		 Allowed values:
		 • 1, on, yes, true
		 • 0, off, no, false

`-i`
 `--sync-marker-retry-interval`
	30 seconds	The number of seconds between attempts to insert the replication synchronization marker into the database. The marker indicates when a total database re-synchronization has been performed with the smsMaster database. Inserting marker requires inputBootstrap to acquire a lock; failure to acquire the lock could result in updateLoader timing out. (Optional.)
		 Allowed values: integers, (any signed number of reasonable value, usually in decimal/octal/hex).

`-h`
 `--help`
	false when missing	Shows the help for this binary.
		 Allowed values:
		 • 1, on, yes, true
		 • 0, off, no, false

`-b`
 `--build-info`
	false when missing	Prints out program build information of the binary.
		 Allowed values:
		 • 1, on, yes, true
		 • 0, off, no, false

Note: Long options can be separated from their values by an equal sign ('='), or you can pass the value as the following argument on the command line (for example, '--port 4000' or '--port=4000'). Short options must have their values passed after them on the command line, and in the case of boolean short options, cannot have values (they default to true) (e.g., '-p 4000' or '-f').

Failure

If inputBootstrap fails, it will send error messages to stdout and syslog.

repConfigWrite

Purpose

This command can be used to create a replication.config file. It reads the local database to obtain the replication set-up and writes the file to the directory specified by the output parameter.

Generally this function is performed using the SMS Java screens.

For more information, see:

- *replication.config File* (on page 38)
- *smsDumpRepConfig* (on page 194)
- *Service Management System User's Guide*

Startup

This task is started by clicking **Create Config File** on the **Table Replication** tab of the Node Management screen.

It can also be started on the SMS from the command line.

For more information about the Node Management screen, see *SMS User's Guide*.

Configuration

repConfigWrite accepts the following command-line options:

Usage:

```
repConfigWrite [-user user/password] [-output file]
```

The available parameters are:

Parameter	Default	Description
`-user`		Oracle user/password for the SMF. **Example:** `smf/smf`
`-output`	./repCofigNNNNN (Where NNNNN is a version number that counts for the number of times the file has generated OK.)	The output path and filename for the replication.config file. (Optional.) **Example:** `/IN/service_packages/SMS/etc/replication.config`

Failure

If repConfigWrite fails, replication.config may not have been written correctly. You can check the content of replication.config with smsDumpRepConfig. If there is a problem with replication.config, replication will not work.

Output

The repConfigWrite writes error messages to the system messages file, and also writes additional output to the specified directory and file.

resyncServer

Purpose

resyncServer initiates resyncs between databases by sending a resync request to a node Master process. This overwrites data in an SCP with data from the SMF. The node Master process is usually smsMaster.

This process is started by the smsMaster when a database resync is required and runs only for the duration of the resync. It should not be run manually.

Configuration

resyncServer accepts the following command line options.

Usage:

```
resyncServer inf_node address port enhanced_recovery
```

The available parameters are:

Parameter	Default	Description
inf_node		The node with the database which will be updated.
address		The ip address or hostname of the node which will be updated.
port		The port number on the node which will be updated.
enhanced_recovery	off	If set to on, the number of rows in the inferior database will not be counted during the resync. **Allowed values:** on, off.

Output

resyncServer writes error messages to the system messages file, and also writes additional output to **/IN/service_packages/SMS/tmp/resyncServer.log**.

setupOracleWallet.sh

Purpose

The **setupOracleWallet.sh** script automatically creates the Oracle server wallet on the SMS by performing a sequence of orapki commands. The Oracle server wallet is the single-sign-on wallet that is used when connecting securely to the database and that contains certificate information for identifying the Oracle server. Use this script only if you are using SSL connections to the database.

For information about creating the Oracle wallet automatically by using **setupOracleWallet.sh**, see *Creating the Oracle Wallet Automatically by Using setupOracleWallet.sh* (on page 62).

Important: You will not need to re-run this script after you complete the Oracle server wallet setup.

Information Required by setupOracleWallet.sh

The following table lists the information that is required by the **setupOracleWallet.sh** script.

Required Item	Description
Oracle wallet base directory	The base directory for the Oracle wallet. Specify the base directory to use for the Oracle root and Oracle server wallets. On a clustered SMS specify a file system that is cluster-wide to allow all instances to access the same wallet information in a single location. On a non-clustered system the default location for the Oracle wallet base directory is: **/u01/app/wallets/oracle/** On a clustered system the default location for the Oracle wallet base directory is: **/global/oracle/app/wallets/oracle/**
ISO country code	The local international country (ISO) code for your country. Specify the two-letter code.
Wallet passwords	The password to use for the root CA wallet and the password to use for the server wallet. You will be prompted for the password each time the wallet is accessed. **Note:** Wallet passwords have length and content validity checks applied to them. Generally passwords should have a minimum length of eight characters and contain alphabetic characters combined with numbers and special characters.

Startup

You run **setupOracleWallet.sh** on the SMS node from the command line. You must be logged in as user *oracle*. If Convergent Charging Controller is installed on a clustered system then you should run **setupOracleWallet.sh** only on the primary SMS node.

Configuration

The **setupOracleWallet.sh** script is configured by the following command-line parameters.

Usage:

```
setupOracleWallet.sh [-k keysize] [-v validdays] [-s server_certificate] [-t
root_certificate] [-w wallet_base]
```

Parameter	Default	Description
-k *keysize*	2048	The keysize for certificate keys.
-v *validdays*	3650	The validity period for certificates in days.
-s *server_certificate*	NA	The signed certificate file for the server wallet; for example, **./server/cert.txt**.
-t *root_certificate*	NA	The root certificate file of the certificate authority (CA); for example, **./root/b64certificate.txt.**

Parameter	Default	Description
-w *wallet_base*	NA	The base directory for the Oracle wallet. If not specified, the script prompts you to enter the location of the Oracle wallet base directory. Choose a directory accessible by user *oracle*; for example, on a non-clustered SMS choose: **/u01/app/wallets/oracle** On a clustered system choose a directory located in a cluster global file system; for example: **/global/oracle/app/wallets/oracle** The script creates the following subdirectories for the root and server wallets under the wallet base location: **./root** and **./server**.

Ways to run setupOracleWallet.sh

When you initially run **setupOracleWallet.sh** you specify whether to use self-signed certificates or certificates signed by a commercial CA. You can optionally specify the `-k`, `-v`, and `-w` command-line parameters; for example:

```
setupOracleWallet.sh -k keysize -v validdays -w wallet_base
```

If you specify to use self-signed certificates then **setupOracleWallet.sh** creates the self-signed root certificate and exports it to the following file:

./root/b64certificate.txt

Where **./root** is a sub-directory of the base directory for the Oracle wallet. You must import this certificate into the Java **lib\security\cacerts** file on each client PC by using the Java keytool utility. See *Adding Trusted Certificates to the Keystore on Client PCs* (on page 62) for more information.

If you specify to use a commercial CA to sign your certificates then **setupOracleWallet.sh** creates the certificate request file that you must send to the commercial CA for signing. When the commercial CA returns the signed certificate, you must rerun **setupOracleWallet.sh** to add the trusted CA certificate and the CA-signed certificate to the server wallet. You can optionally specify the `-s` and `-t` command-line parameters; for example:

```
setupOracleWallet.sh -s server_certificate -t root_certificate
```

smsCompareResyncClient

Purpose

This is a child process of updateLoader. It is called by smsCompareResyncServer and updates the SCP during replication on a clustered install. It also performs database resynchronizations and comparisons on the inferior node during replication checks.

This process is not intended to be be started manually. It is installed on the SLC.

Configuration

smsCompareResyncClient accepts the following command line options.

Usage:

```
smsCompareResyncClient -n int [-u usr/pwd] [-h] [-b] [-i int] [-p port] [-o seconds]
[-t] [--outside-throttle-sample-rate int] [--inside-throttle-sample-rate int] [--
start-threshold int] [--stop-threshold int] [-s dir] [--database-write-buffer-size
int] [--database-read-buffer-size int] [--database-commit-period int] [--max-buffer-
size int] [--dump-core-instead-of-exception] [--long-raw-size=max_size]
```

The available parameters are:

Parameter	Default	Description
`-n` `--node-id`		The node number of the client. (Required.) **Allowed values:** integer (any signed number of reasonable value, usually in decimal/octal/hex)
`-u` `--oracle-user`	smf/smf	The Oracle database connection string. (Optional.) **Allowed values:** string
`-h` `--help`	false	Prints this help screen. (Optional.) **Allowed values:** • 1, on, yes, true • 0, off, no, false
`-b` `--build-info`	false	Prints out program build information then exits. (Optional.) **Allowed values:** • 1, on, yes, true • 0, off, no, false
`-i` `--inform-parent`	no process is informed	Process to inform when the process is in a position to resync/compare. For use only when used from the updateLoader process. (Optional.) **Allowed values:** integers (any signed number of reasonable value, usually in decimal/octal/hex)
`-p` `--port`		The port to listen on for connections from the server. (Optional.) **Allowed values:** integers (any signed number of reasonable value, usually in decimal/octal/hex)
`-o` `--tcp-timeout`	-1	Timeout (in seconds) on TCP connection. (Optional.) **Allowed values:** • positive integers • -1 = no timeout
`-t` `--throttle-cpu`	false	To throttle the client. (Optional.) **Allowed values:** • 1, on, yes, true • 0, off, no, false
`--outside-throttle-sample-rate`		Sample rate in seconds for throttling while not throttling. (Required if -t given.) **Allowed values:** integers (any signed number of reasonable value, usually in decimal/octal/hex)
`--inside-throttle-sample-rate`		Sample rate in seconds for throttling while throttling. (Required if -t given.) **Allowed values:** integers (any signed number of reasonable value, usually in decimal/octal/hex)

Parameter	Default	Description
`--start-threshold`		CPU usage percentage to start throttling at. (Required if -t given.) **Allowed values:** integers (any signed number of reasonable value, usually in decimal/octal/hex)
`--stop-threshold`		CPU usage percentage to end throttling at. (Required if -t given.) **Allowed values:** integers (any signed number of reasonable value, usually in decimal/octal/hex)
`-s` `--storage-dir`	/tmp	Directory to store required database changes in during a resync. (Optional.) **Allowed values:** string
`--database-write-buffer-size`	10	The size in terms of records of the buffer size for writing to the database. (Optional.) **Allowed values:** integers (any signed number of reasonable value, usually in decimal/octal/hex)
`--database-read-buffer-size`	100	The size in terms of records of the buffer size for reading to the database. (Optional.) **Allowed values:** integers (any signed number of reasonable value, usually in decimal/octal/hex)
`--database-commit-period`	1000	The number of changes to make to the database before committing them to the database. (Optional.) **Allowed values:** integers (any signed number of reasonable value, usually in decimal/octal/hex)
`--max-buffer-size`	approximately 50Mb	The maximum size (in bytes) to allow messages received from the server to be. This size is reflected in the maximum size of bytes to allocate in memory for such messages. (Optional.) **Allowed values:** integers (any signed number of reasonable value, usually in decimal/octal/hex)
`--dump-core-instead-of-exception`	false	If set, will force the process to dump a core file (and exit) if any network messages are received bigger than max-buffer-size. **Allowed values:** • 1, on, yes, true • 0, off, no, false
`--long-raw-size`	512K	The maximum size (in bytes) to allocate for a long raw field used in a resync. **Allowed values:** • 512K.

Note: Long options can be separated from their values by an equal sign ('='), or you can pass the value as the following argument on the command line (for example, '--node-id 257' or '--node-id=257'). Short options must have their values passed after them on the command line, and in the case of boolean short options, cannot have values (they default to true) (for example, '-p 4000' or '-t').

smsCompareResyncServer

Purpose

smsCompareResyncServer performs comparisons and resyncs of data in specified tables and replication groups. This enables you to:

- Check that replication is working correctly
- Force updates of data between nodes

Note: This process is usually started by resyncServer.

smsCompareResyncServer is installed on the SMS.

Configuration

smsCompareResyncServer accepts the following command line options.

Usage:

```
smsCompareResyncServer [--dump-core-instead-of-exception] --max-buffer-size=size [--
dont-count-rows] [--inform-master] [--database-read-buffer-size=size] [--cancel-on-
eof] [--use-ip=int] [--report-directory=base_dir] [--tcp-timeout] [--input-
file=config_file_name] [--oracle-user=user] [--build-info] [--help] [--long-raw-
size=max_size]
```

The available parameters are:

Parameter	Default	Description
`--dump-core-instead-of-exception`	false when missing	If set, will force the process to dump a core file (and exit) if any network messages are received bigger than max-buffer-size. **Allowed values:** • 1, on, yes, true • 0, off, no, false
`--max-buffer-size`	approximately 50Mb	The maximum size (in bytes) to allow messages sent to the client to be. This size is reflected in the maximum size of bytes to allocate in memory for such messages. **Allowed values:** integers (any signed number of reasonable value, usually in decimal/octal/hex).
`-d` `--dont-count-rows`	false when missing	Do not make a count of the rows in the database. For very large comparisons/resyncs this may give a speed improvement. **Allowed values:** • 1, on, yes, true • 0, off, no, false
`-m` `--inform-master`	false when missing	If performing a resync, inform the smsMaster. **Allowed values:** • 1, on, yes, true • 0, off, no, false

Parameter	Default	Description
`--database-read-buffer-size`	100	The size (in records) of the buffer size for reading from the database. Must match the --database-write-buffer-size specified for the smsCompResyncClient. (Optional.) **Allowed values:** integers (any signed number of reasonable value, usually in decimal/octal/hex). **Note:** Performance of the compare/resync can be seriously impacted if the number specified is too low, a recommended value for this parameter is 1000+.
`--cancel-on-eof`	false when missing	Tells the server to cancel the resync/compare when EOF on standard input is reached. **Note:** This option is specifically for the inetCompareServer setup. **Allowed values:** • 1, on, yes, true • 0, off, no, false
`--use-ip`	none	Forces the server to use the IP address ranked as per the mentioned integer for each client. If there are not enough IP addresses listed, or this option is not specified, it will start from the first IP address, attempting each in turn until connected. **Allowed values:** integers (any signed number of reasonable value, usually in decimal/octal/hex). **Examples:** If `--use-ip = 2`, the server will use the second IP address listed.
`-r` `--report-directory`	/IN/html/output/SMS/	The base directory to create and store final reports in. (Optional.) **Allowed values:** string.
`-o` `--tcp-timeout`	0	Timeout (in seconds) on TCP connection. (Optional.) Zero = no timeout. **Allowed values:** integers (any signed number of reasonable value, usually in decimal/octal/hex).
`-i` `--input-file`	Input is expected on the standard input stream.	Contains the name of a configuration file as input information for performing a resync/compare. (Optional.) See Input file for details. **Allowed values:** string
`-u` `--oracle-user`	smf/smf	The Oracle database connection string. (Optional.) **Allowed values:** string
`-b` `--build-info`	false when missing	Prints out program build information then exits. **Allowed values:** • 1, on, yes, true • 0, off, no, false

Parameter	Default	Description
`-h` `--help`	false when missing	Prints this help screen. **Allowed values:** • 1, on, yes, true • 0, off, no, false
`--long-raw-size`	512K	The maximum size (in bytes) to allocate for a long raw field used in a resync. **Allowed values:** • 512K.

Notes:

- All options apart from '-h', '-b' and '-i' can be specified in the input configuration.
- Long options can be separated from their values by an equal sign ('='), or you can pass the value as the following argument on the command line (for example, '--tcp-timeout 100' or '--tcp-timeout=100'). Short options must have their values passed after them on the command line, and in the case of boolean short options, cannot have values (they default to true) (for example, '-o 100' or '-h').

Input file

These are the configuration parameters contained within the input file optionally used by smsCompareResyncServer.

`Replication`

Syntax:	`Replication = {`*`list_of_replication_parameters`*`}`
Description:	The Replication section lists the required replication parameters.
Type:	List
Optionality:	Required
Allowed:	
Default:	none
Notes:	
Example:	

`perform`

Syntax:	`perform = "`*`action`*`"`
Description:	The action to be taken by smsCompareResyncServer.
Type:	Boolean
Optionality:	Required
Allowed:	• resync • compare
Default:	none
Notes:	
Example:	`perform = "resync"`

`report-row-number-limit`

Syntax:	`report-row-number-limit = `*`max_value`*
Description:	For a comparison, the maximum number of differences in a group that will be reported. For a resync, the maximum number of errors to report of each group synchronized.
Type:	Integer
Optionality:	Optional (default used if not set).
Allowed:	
Default:	-1
Notes:	Specifying -1 indicates no limit will be imposed.
Example:	`report-row-number-limit = 100`

`produce-final-reports`

Syntax:	`produce-final-reports` = true|false
Description:	Whether or not to produce final reports.
Type:	Boolean
Optionality:	Optional (default used if not set).
Allowed:	true, false
Default:	true (reports produced)
Notes:	
Example:	`produce-final-reports = false`

`report-directory`

Syntax:	`report-directory = "`*`dir`*`"`
Description:	Specify which directory reports are to be written to.
Type:	String
Optionality:	Optional (default used if not set)
Allowed:	Any valid directory
Default:	/IN/html/output/SMS
Notes:	
Example:	`report-directory = "."`

`report-after`

Syntax:

```
report-after = {
  type = "[comparisons|seconds]"
  count = number
}
```

Description:	Depending on the type option: Specify how many comparisons the client should process before sending progress report to the server. Specify how many seconds the client should allow to elapse before sending a progress report to the server.
Type:	List

Optionality:	Optional (default used if not set)
Allowed:	
Default:	No progress reports are provided.
Notes:	**type** and **count** are both mandatory when report-after is specified.
Example:	`report-after = {` `count = 10` `type = "seconds"` `}`

```
report-after = {
  count = 10
  type = "seconds"
}
```

stop-on-limit

Syntax:	`stop-on-limit = `*`true`*`|`*`false`*
Description:	A flag to tell the server to stop a resync or comparison for a replication group after the *report-row-number-limit* (on page 187) is reached.
Type:	Boolean
Optionality:	Optional (default used if not set).
Allowed:	true, false
Default:	false (do not stop)
Notes:	
Example:	`stop-on-limit = false`

view

Syntax:	`view = {}`
Description:	Describes the nodes, tables and groups for the replication action.
Type:	List
Optionality:	Required
Allowed:	
Default:	none
Notes:	
Example:	

Node

Syntax:

```
Node {
    id = number
    address = [
        "string", "string", ...
    ]
}
```

Description:	A list of nodes for the replication action to work on.
Type:	Array
Optionality:	Required
Allowed:	
Default:	none
Notes:	
Example:	

`id`

Syntax:	`id = `*`id`*
Description:	The ID of the node to be used.
Type:	Integer
Optionality:	Optional (default used if not set)
Allowed:	
Default:	Values from replication node configuration
Notes:	For normal replication resyncronization, these are calculated using the replication node configuration. When running from the command line this must match the --node-id of a smsCompareResyncClient.
Example:	`id = 301`

`address`

Syntax:

```
address = [
        "string", "string", ...
    ]
```

Description:	An array of IP addresses and port numbers for this node ID.
Type:	Array
Optionality:	Required
Allowed:	
Default:	None
Notes:	Internet Protocol version 6 (IPv6) addresses must be enclosed in square brackets []; for example: **`[2001:db8:`***`n`*`:`*`n`*`:`*`n`*`:`*`n`*`:`*`n`***`]`** where *n* is a group of 4 hexadecimal digits. The industry standard for omitting zeros is also allowed when specifying IP addresses. Use a comma to separate address entries or specify each entry on a separate line.

Example:

```
address = [
    "192.0.2.1:4000"
    "[2001:db8:0000:1050:0005:0600:300c:326b]:3004"
    "[2001:db8:0:0:0:500:300a:326f]:1234:SMF"
    "[2001:db8::c3]:1234:SMF"
    ]
```

`tables`

Syntax:

```
tables = [
    {
        table = "string" [
            groups-cover-table-on-scp = bool
        ]
        key-columns = [
            "str", "str", ...
        ]
        other-columns = [
            "str", "str", ...
        ]
    }
]
```

Description:	An array of tables to action for this node ID.
Type:	Array

Optionality:	Required
Allowed:	
Default:	none
Notes:	
Example:	

`table`

Syntax:	`table = "str"`
Description:	The name of the table to be used in this operation.
Type:	String
Optionality:	Required
Allowed:	
Default:	none
Notes:	
Example:	`table = "TEST_REP"`

`groups-cover-table-on-scp`

Syntax:	`groups-cover-table-on-scp = true|false`
Description:	
Type:	Boolean
Optionality:	Optional (default used if not set)
Allowed:	true, false
Default:	false
Notes:	
Example:	`groups-cover-table-on-scp = true`

`key-columns`

Syntax:

```
key-columns = [
    "str", "str", ...
]
```

Description:	An array of the table column keys which are to be used for this table and this operation.
Type:	Array
Optionality:	Optional (default used if not set).
Allowed:	Any valid key column name for this table.
Default:	Pre-configured values.
Notes:	Normal replication resyncs are pre-configured and restricted to a maximum of 3 keys. When running from the command line this restriction is lifted. These columns must exist on the remote platform.

Example:

```
key-columns = [
     "NUMBER_3"
]
```

other-columns

Syntax:	`other-columns = [ "str", "str", ... ]`
Description:	An array of the non keyed columns which are to be used for this table and this operation.
Type:	Array
Optionality:	Optional (default used if not set).
Allowed:	Any valid non keyed column name for this table.
Default:	Pre-configured values.
Notes:	For normally replication resyncs these are pre-configured. These columns must exist on the remote platform.
Example:	`other-columns = [ "VARCHAR_1","LONG_RAW_2","CHAR_4", "DATE_5" ]`

Groups

Syntax:

```
groups = [
    {
        group = <str>
        table = <str>
        ranges = [
            <rangeData>, <rangeData>, ...
        ]
        nodes = [
            <int>, <int>, ...
        ]
    }
 ]
```

Description:	An array of groups associated with this operation.
Type:	Array
Optionality:	Required
Allowed:	
Default:	none
Notes:	
Example:	

group

Syntax:	`group = "str"`
Description:	The name associated with a node.
Type:	String
Optionality:	Required
Allowed:	Any character
Default:	none
Notes:	
Example:	`group = "TEST_REP_0"`

table

Syntax:	`table = "`*`str`*`"`
Description:	The name of the table associated with this group.
Type:	String
Optionality:	Required
Allowed:	Any characters
Default:	none
Notes:	
Example:	`table = "TEST_REP"`

ranges

Syntax:

```
ranges = [
        {
            from = [<from data>]
            to = [<to data>]
        }
    ]
```

Description:	An array specifying the ranges to be associated with this group, table, and for this node.
Type:	Array
Optionality:	Required
Allowed:	
Default:	none
Notes:	Although an index to support the ranges specified is not required, it is recommended an index is used for performance reasons. The number of elements in the from and to conditions must be the same and must match tables, key-columns entry for the table specified.

Example:

```
ranges = [
        {
            from = [
                "1"
            ]
            to = [
                "2"
            ]
        }
    ]
```

nodes

Syntax:	`nodes = [ `*`int`*`, `*`int`*`, [..]]`
Description:	The list of nodes to be associated with this group.
Type:	Array
Optionality:	Required
Allowed:	Any nodes defined in Node section.
Default:	none
Notes:	These nodes must have been defined already in the Node section.

Example:

```
nodes = [
        301
    ]
```

Input file example

This is an example of what the input configuration will look like, the indentation format is for readability.

```
replication = {
    perform = "resync"
    report-row-number-limit = 100
    produce-final-reports = true
    report-directory = "."
    report-after = {
        count = 10
        type = "seconds"
    }
    stop-on-limit = false
}
view = {
    nodes = [
    {
        id = 400
        address = [
            "127.0.0.1"
        ]
    }
    ]
    tables = [
    {
        table = "TEST_REP"
        groups-cover-table-on-scp = true
        key-columns = [
            "NUMBER_3"
        ]
        other-columns = [
            "VARCHAR_1",
            "LONG_RAW_2",
            "CHAR_4",
            "DATE_5"
        ]
    }
    ]
    groups = [
    {
        group = "TEST_REP_0"
        table = "TEST_REP"
        ranges = [
            {
                from = [
                    "1"
                ]
                to = [
                    "2"
                ]
            }
        ]
        nodes = [
            400
        ]
    }
    ]
}
```

Output

smsCompareResyncServer writes error messages to the system messages file.

smsCompareResyncServer writes replication checks and database comparisons to the **/IN/html/output/SMS/compare/***inferior_ node_numberl* directory.

smsDumpRepConfig

Purpose

smsDumpRepConfig parses and displays the contents of **replication.config**. This provides access to the contents of the binary file where the replication configuration data is held.

For more information, see *replication.config File* (on page 38).

Configuration

The smsDumpRepConfig supports the following command-line options:

Usage:

```
smsDumpRepConfig -f filename [-v]
```

The available parameters are:

Parameter	Default	Description
-f	/IN/service_packages/SMS/etc/replication.config	Location of the configuration file to be displayed.
-v		Verbose, displays extra information, including column and field names.

Failure

If smsDumpRepConfig fails, it will send error messages to stdout and syslog. If an error is displayed while parsing a r**eplication.config** file, the file may be corrupted.

Output

smsDumpRepConfig displays output to stdout.

Example:

This text is an example of the output from a simple replication.config file which includes SMS and ACS replication groups between nodes 1 and 301.

```
smsDumpRepConfig: File /IN/service_packages/SMS/etc/replication.config
smsDumpRepConfig: (PAD = 0)
smsDumpRepConfig: Short listing.  Use -v (verbose) for full listing
--------------------------------------------------------------------------------
smsDumpRepConfig: Table, Column, Group definitions...
--------------------------------------------------------------------------------
TABLE [ACS_CALL_PLAN]
TABLE [ACS_CALL_PLAN_PROFILE]
TABLE [ACS_CALL_PLAN_STRUCTURE]
TABLE [ACS_CLI_CALL_PLAN_ACTIVATION]
TABLE [ACS_CUSTOMER]
TABLE [ACS_CUSTOMER_CLI]
TABLE [ACS_CUSTOMER_SN]
TABLE [ACS_FN_TYPE]
TABLE [ACS_GLOBAL_PROFILE]
TABLE [ACS_LANGUAGE]
TABLE [ACS_NETWORK_KEY]
TABLE [ACS_SN_CALL_PLAN_ACTIVATION]
TABLE [SMF_ALARM_MESSAGE]
TABLE [SMF_STATISTICS]
TABLE [SMF_STATISTICS_DEFN]
------------------------------------------------------------------------------
smsDumpRepConfig: Replication Groups configured for each node...
------------------------------------------------------------------------------
NODE NUMBER [1] Prim (192.168.0.144) Sec (0.0.0.0)
NODE NUMBER [301] Prim (192.168.0.142) Sec (0.0.0.0)
    GROUP [ACS_CUSTOMER] [Prim=-1] Min=('+0','',') Max=('+9','','')
    GROUP [ACS_FN_TYPE] [Prim=-1] Min=('+0','',') Max=('+9','','')
    GROUP [ACS_CALL_PLAN_PROFILE] [Prim=-1] Min=('+0','',') Max=('+9','','')
    GROUP [ACS_CALL_PLAN_STRUCTURE] [Prim=-1] Min=('+0','',') Max=('+9','','')
    GROUP [ACS_CALL_PLAN] [Prim=-1] Min=('+0','',') Max=('+9','','')
    GROUP [ACS_CUSTOMER_CLI] [Prim=-1] Min=('+0','',') Max=('+9','','')
    GROUP [ACS_CUSTOMER_SN] [Prim=-1] Min=('+0','',') Max=('+9','','')
    GROUP [SMF_STATISTICS_DEFN] [Prim=-1] Min=('!','!',') Max=('~','~','')
    GROUP [ACS_CLI_CALL_PLAN_ACTIVATION] [Prim=-1] Min=('+0','',') Max=('+9','','')
    GROUP [ACS_GLOBAL_PROFILE] [Prim=-1] Min=('+0','',') Max=('+9','','')
    GROUP [ACS_LANGUAGE] [Prim=-1] Min=('+0','',') Max=('+9','','')
    GROUP [ACS_NETWORK_KEY] [Prim=-1] Min=('+0','',') Max=('+9','','')
    GROUP [ACS_SN_CALL_PLAN_ACTIVATION] [Prim=-1] Min=('+0','',') Max=('+9','','')
```

Note: Both nodes are primaries for their groups and have no secondary network address configured.

smsIorDump

Purpose

The smsIorDump utility enables you to display details about the IORs (Interoperable Object References) available to CORBA services. You use smsIorDump to investigate java exceptions related to CORBA.

The smsIorDump utility is located in the following directory:

/IN/service_packages/SMS/bin/

Configuration

The smsIorDump utility supports the following command-line options:

Usage:

```
smsIorDump [-u user/pw] -i IOR_str
```

Where:

- *user/pw* is the user and password for the database on the SMS
- *IOR_str* is the IOR whose details you want to display

Fields Displayed by smsIorDump

This table describes the fields that display when you run the smsIorDump utility.

Field	Description
NAME	(Always display) The unique CORBA server label agreed between the server and the client.

Field	Description
CLASS	(Display if available) Provides version information.
IP	(Display if available) The IP address that provides a key for distinguishing between multiple servers of the same type. Note that this IP address is not used to locate the server.
IOR	(Always display) The CORBA Interoperable Object Reference (IOR) that specifies the class of object actually served, and the host and port number of the server. The host and port number define the address that will be used to locate the server at run-time.

smsLogTest

Purpose

smsLogTest generates an alarm and writes it to the syslog on the local machine. You can configure the alarm details.

Configuration

smsLogTest supports the following command-line options:

Usage:

```
./smsLogTest name severity message [copies] [alarm_type_id]
```

The available parameters are:

Parameter	Default	Description
name	none	The subsystem name/identifier. (Required.)
severity	none	The severity value. (Required.) **Allowed values:** **N** or **0** [Notice] **W** or **1** [Warning] **E** or **2** [Error] **C** or **3** [Critical] **-** or **4** [clear]
message	none	The message to log. (Required.)
[copies]	1	Number of times to generate the alarm. (Optional.) **Allowed values:** integer
[{alarm_type_id}]	none	Optional Alarm Type ID associated with the message. This must be up to a 9-digit number between braces (for example, {123456789}

Examples:

```
./smsLogTest smsAlarmRelay %d \"Failed to connect to Oracle\" 4\n {123456789}
./smsLogTest smsMaster %d \"Startup Successful\"
```

Failure

If smsLogTest fails, no alarm will be generated.

Output

smsLogTest displays progress and errors to stdout. It writes the alarm output to the local syslog.

smsManualRequester

Purpose

smsManualRequester sends update requests to the smsMaster.

Configuration

smsManualRequester supports the following command-line options:

Usage:

```
smsManualRequester [-nodeid value]
```

The available parameters are:

Parameter	Default	Description
value		the ID of the node (Optional)

Output

smsManualRequester displays the output to local terminal.

Examples:

```
./smsManualRequester -nodeid 999
Nov 22 22:01:48 smsManualRequester(6578) NOTICE: Update Requester
`./smsManualRequester' process registered (node 999)
Enter table name (start with `-' to indicate delete)
?-to set info return:-ACS_CUSTOMER
Enter key names (key1,key2...):id
Enter values for keys & columns(terminate with ##):29
Enter values for keys & columns(terminate with ##):##
initialiseNode: Reading '/IN/service_packages/SMS/etc/replication.def'
initialiseNode: heartbeatPeriod 20
initialiseNode: heartbeatTimeout 20
initialiseNode: connectionTimeout 2
initialiseNode: masterPortNum 12343
initialiseNode: queueWarnThresh 5
initialiseNode: queueErrThresh 100000
initialiseNode: queueCritThresh 1000000
initialiseNode: hBTolerance 10.0
initialiseNode: commitIdleTime 0.100000
initialiseNode: commitBusyTime 10.0
initialiseNode: tcpAbortSecs 20
initialiseNode: oracleUserPass '/'
initialiseNode: reportDir '/IN/service_packages/SMS/tmp/'
initialiseNode: statusFile '/IN/html/status.html'
initialiseNode: configFilePath '/IN/service_packages/SMS/etc/replication.config'
initialiseNode: configFileName 'replication.config'
initialiseNode: node number 999
initialiseNode: node type 3
initialiseNode: s side updates 1
Nov 22 22:02:17 smsManualRequester(6578) NOTICE: Reached master node 1 at
`192.168.0.198'
Enter table name (start with `-' to indicate delete)
```

```
?-to set info return:
.....
```

smsProcessCdr

Purpose

smsProcessCdr processes EDRs based on the rules set in a format file. The format file describes the fields, literal strings and functions to apply to the input data in order to produce the desired output EDR.

Functions include:

- Field selection
- Reordering
- Delimiter specification
- String concatenation with static strings and field values (such as would be required for field #13 in the EDR SRS)
- Multi-level pattern matching (as would be required for field #1) conditional field selection (as would be required for field #11)

This process is typically used to perform the following tasks for EDR files and ACS PIN log files:

1. (Optional) format conversion on files.
2. Move of files to a medium-term archive area.
3. (Optional) copy of files to directory for external retrieval.
4. Cleanup of expired files from the archive area.

Specification and implementation of EDR processing requirements is typically a system integration task which is performed prior to final system acceptance. This is usually implemented by the shell script **smsCdrProcess.sh**.

To prevent the EDR from being processed, see *Configuring smsCdrProcess.sh* (on page 115).

Configuration

smsProcessCdr accepts the following command line parameters.

Usage:

```
smsProcessCdr [-t edr_format] -d in_dir -D out_dir [-s in_suffix] [-p in_prefix] [-S
out_suffix] [-P out_prefix] [-u usr/pwd] [-l tz] [-h] -b
```

The available parameters are:

Parameter	Default	Description
-t *edr_format*	none	The filename of the EDR format file. **Note:** No format conversion is performed by default. The formatting file supports the following: • Fields • Literal strings • Functions
-d *in_dir*	none	The directory to read EDRs from.
-D *out_dir*	none	The directory to write processed EDRs to.
-s *in_suffix*	none	The suffix that input EDR files must match (these are stripped from the output file name).

Parameter	Default	Description
`-p` *in_prefix*	none	The prefix that input EDR files must match (these are stripped from the output file name).
`-S` *out_suffix*	none	The suffix to add to output EDR files.
`-P` *out_prefix*	none	The prefix to add to output EDR files.
`-u` *usr/pwd*	/	The Oracle user and password to use. **Note:** smsProcessCdr will only attempt to connect to the database if the EDR format file contains functions that require it.
`-l` *tz*	none	Alternate timezone TCS and TCE EDR fields are converted to.
`-h`	none	Displays a help page.
`-b`	none	Allows blank header tag values. Treats non-existent header tags as blank value.

Example 1

The following command would:

- Use **/IN/service_packages/SMS/bin/cdrFormat.fmt** as theEDR format file
- Process every file matching the pattern **/IN/service_packages/SMS/cdr/received/scp2_acs*.cdr**

```
smsProcessCdr -t /IN/service_packages/SMS/bin/cdrFormat.fmt -d
/IN/service_packages/SMS/cdr/received -D /tmp/processedCdrs -p scp2_acs -s .cdr -P
ACS_ -S .out -u smf/smf
```

The output file name is a transformation of the input file name. For example, with the parameters supplied above, an input file **/IN/service_packages/SMS/cdr/received/scp2_acs20010831120012.cdr** would have output file named **/tmp/processedCdrs/ACS_20010831120012.out**.

Example 2

The following command:

```
smsProcessCdr -t /IN/service_packages/SMS/bin/mobifone.fmt -d
/IN/service_packages/SMS/cdr/received -D /tmp/processedCdrs -p scp2_acs -s .cdr -P
ACS_ -S .out -u smf/smf
```

Would cause the following file to be parsed as the EDR format file **/IN/service_packages/SMS/bin/mobifone.fmt**.

After parsing is complete, the binary will process its input files. With the parameters supplied above, this would be every file matching the pattern:

/IN/service_packages/SMS/cdr/received/scp2_acs*.cdr

When an input EDR is successfully processed, it is written out to an output EDR file. One output EDR file is created for each input EDR file. The output file name is a transformation of the input file name.

The input file **/IN/service_packages/SMS/cdr/received/scp2_acs20010831120012.cdr** would produce the following output file **/tmp/processedCdrs/ACS_20010831120012.out**.

Format File configuration

A EDR format file consists of field specifiers which translate input data to an output format.

The valid field specifiers are:

- Header tags
- Standard fields
- Special fields
- Strings
- Functions
- Format characters

Header tags

A HEADER tag may be specified in the format file passed in providing a single header line at the top of processed output files.

The header appears once and contains all HEADER tag values concatenated and space separated.

Through the use of a `-b` option passed in to smsProcessCdr at runtime blank values are allowed. When any tags are missing their respective value will be set blank rather than exiting on error. Underscores are allowed by default with no extra settings.

Standard fields

Standard fields are fields which relate to tags in the input EDR.

ACS EDRs have the following format:

```
APPLICATION|tag1=value1|tag2=value2| . . . |tagn=valuen
```

A standard field is any one of the tag values.

If a EDR File format contains a field tag, then the corresponding value from the input EDR, value, will be written to the output EDR. Standard tags are written into the EDR format file using the same text which is used to specify them in the input EDR.

Special fields

Special fields are for data extracted from a EDR which does not occur in the input as a *tag*=*value* pair.

The only example of this is the EDR application name field, which always occurs as the first element in a EDR.

Placing the field `<APPLICATION>` in the EDR format file will cause the application name from the input EDR to be written to the output EDR.

Strings

Strings are used to write literal text to the output EDR.

Strings appear in the EDR format file as double quoted strings "*data*".

Any characters occurring in *data* are written, verbatim, to the output EDR file. This can be used to supply field delimiters (for example: ",") or to hard code output values (for example: "0,2,-1,").

Functions

Functions are programmatic transformations that can be applied to values.

Functions occur in the DR format file with the form:

```
function_name ( function_parameters )
```

Functions always produce a textual output.

The format of the functions used in smsProcessCdr are the same or similar to those used in the LISP programming language.

Most functions will support expressions as parameters so long as they produce textual output. The following types are included:

- Standard fields
- Special fields
- Strings
- Functions

Boolean expressions

Boolean expressions are used as parameters to COND functions (and could be used as parameters to other functions at a later date).

Boolean functions are functions which evaluate to either TRUE or FALSE. The compiler will not allow Boolean functions to be used as 'top level' functions, but they may be nested in other functions to provide the ability to test conditions.

EQUALS function

The EQUALS function compares two expressions for equality. EQUALS evaluates to TRUE if the expressions are equivalent. Otherwise it evaluates to FALSE. The equality test is a case-sensitive string comparison.

```
EQUALS ( expr1 ,expr2 )
```

Example: This example shows the EQUALS function being used as part of a COND function. EQUALS is used to check whether the application name is "ACS".

```
COND ( ( EQUALS ( APPLICATION , "ACS" ) , "ACS Service" ) , ( TRUE , "Unknown
Service" ) )
```

PREFIX function

The PREFIX function evaluates to TRUE if expr2 is a prefix of expr1. Otherwise it evaluates to FALSE.

```
PREFIX ( expr1 , expr2 )
```

Example: This example shows the PREFIX function being used as part of a COND function. PREFIX is used to check whether the service number (SN) in the input EDR starts with either the digits 0800 or 0900.

```
COND ( ( PREFIX ( SN , "0800" ) , "Freephone" ) , ( PREFIX ( SN , "0900" ) , "Pay
Service" ) , ( TRUE , "Unknown Service" ) )
```

CONCAT function

The CONCAT function concatenates one or more expressions.

```
CONCAT ( expr1 [ , expr2 , expr3 . . . exprn ] )
```

Example: This example concatenates the literal string T0 onto the value of the special field, *APPLICATION*. Therefore, if *APPLICATION* evaluates to CCS then the example would produce the output: T0CCS.

```
CONCAT ( "T0" , APPLICATION )
```

Example: This example shows literal strings being concatenated to a field value and the result of another expression. If the value of the field XYZ is 21 and CCET is 12.32 then the result of the example would be: ABC2112.

```
CONCAT ( "ABC" , XYZ , ROUND ( CCET ) )
```

COND function

The COND function evaluates to an expression on the basis of a series of one or more test Boolean expressions.

The first Boolean expression which evaluates to TRUE causes its associated result expression to be evaluated as the result of the COND function. If none of the Boolean expressions evaluates to TRUE then the result of the COND function is an empty string.

```
COND ( ( bexpr1 , expr1 ) [ , ( bexpr2 , expr2 ) . . . , ( bexprn , exprn ) ] )
```

Examples:

In either of these two examples, the function evaluates to:

- Pizza Shed if the service number is 0800101101
- Pay Service if the service number starts with 0900
- Unknown in all other cases

```
COND ( ( EQUALS ( SN , "0800101101" ) , "Pizza Shed" ) , ( TRUE , ( COND ( ( PREFIX
( SN , "0900" ) , "Pay Service" ) , ( TRUE , "Unknown" )
COND ( ( EQUALS ( SN , "0800101101" ) , "Pizza Shed" ) , ( PREFIX ( SN , "0900" ) ,
"Pay Service" ) , ( TRUE , "Unknown" ) )
```

For more examples, see the examples for PREFIX and EQUALS.

LANGUAGEID function

The LANGUAGEID function evaluates to the ID of a named language (from the ACS_SE_LANGUAGE table). The language name check is case insensitive.

If the named language can not be found, LANGUAGEID evaluates to –1.

```
LANGUAGEID ( "string" )
```

Note: Using the LANGUAGEID function requires a connection to the database, which requires setting the oracle user / password option when invoking smsProcessCdr (unless the default "/" will suffice).

Example: This example checks the language ID from the input EDR (the LGID field). If the LGID is the same as the ID for the language English, then the expression evaluates to 1. If it is French, it evaluates to 2, if it is German, it evaluates to 3.

```
COND ( ( EQUALS ( LANGUAGEID ( "English" , LGID ) ) , "1" ) , ( EQUALS ( LANGUAGEID
( "French"  , LGID ) ) , "2" ) , ( EQUALS ( LANGUAGEID ( "German" ,  LGID ) ) , "3"
) )
```

ROUND function

The ROUND function interprets the supplied expression as a floating point number and replaces it with the same value rounded to the nearest integer. ROUND also works for negative numbers (using the minus symbol). If the supplied expression cannot be interpreted as a floating point number, then ROUND will evaluate to 0.

```
ROUND ( expr )
```

Examples:

This example evaluates to 2.

```
ROUND ( "2.1" )
```

This example evaluates to 3.

```
ROUND ( "2.6" )
```

If CCET evaluates to 12.3, this example evaluates to 12.

```
ROUND ( CCET )
```

SUBSTR function

The SUBSTR function extracts a substring from a given expression. The parameters:

- *expr* is the source expression
- *start* is an integer indicating where the substring should start (counting starts at zero)
- *length* is an integer indicating how many characters should be read.

If *start* is greater than the length of *expr*, then SUBSTR returns an empty string. If the specified *start* and *length* would cause SUBSTR to read off the end of the input expression, then SUBSTR returns the maximum number of characters it could read.

```
SUBSTR ( expr , start , length )
```

Examples:

This example evaluates to "the".

```
SUBSTR ( "the happy elephant" , 0, 3 )
```

This example evaluates to "e ha".

```
SUBSTR ( "the happy elephant" , 2, 4 )
```

This example evaluates to an empty string.

```
SUBSTR ( "the happy elephant" , 50, 4 )
```

This example evaluates to "appy elephant".

```
SUBSTR (  "the happy elephant" , 5, 40 )
```

Format characters

The format characters are a subset of the ASCII escape characters which allow special characters to be inserted into the output file. This table describes the supported format characters.

Format character	Definition
\n	New line
\r	Carriage return
\t	Tab
\0	Null

File Format example 1

A simple example which just picks up the application name, service number, CLI.

Fields are comma delimited, records are terminated with a newline character (\n).

Format File:

```
<APPLICATION> "," SN "," CLI \n
```

Input CDR file contents:

```
ACS|SN=0800101101|CLI=044784333|XYZ=123
ACS|SN=0900222333|CLI=044784333|XYZ=123
ACS|SN=|CLI=044784333|XYZ=123
```

Output file contents:

```
ACS,0800101101,044784333
ACS,0900222333,044784333
ACS,,044784333
```

File Format example 2

A more complicated example using comments, special fields, and a function.

Fields are space delimited, records are terminated with a newline and a carriage return character.

Format File:

```
// our CDR format file
APPLICATION " "
// fields 2 and 3 are hard coded to be zero
```

```
"0 0 "
ROUND ( CCET ) " "
COND ( (EQUALS(APPLICATION, "CCS"), CONCAT("00", CA)), (TRUE, CONCAT("00", TN)) )
// end of line indicator:
\n \r
```

Input CDR file contents:

```
CCS|XYZ=123|CCET=0.2|TN=123123|CA=321321|ABC=333
ACS|XYZ=123|CCET=8.8|TN=123123|CA=321321|ABC=333
VPN|XYZ=123|CCET=-1.6|TN=123123|CA=321321|ABC=333
CCS|XYZ=123|CCET=BOB|TN=123123|CA=321321|ABC=333
```

Output file contents:

```
CCS 0 0 0 00321321
ACS 0 0 9 00123123
VPN 0 0 -2 00123123
CCS 0 0 0 00321321
```

File Format example 3

Another complicated example using a header, comments, special fields, and a function.

Fields are space delimited, records are terminated with a newline and a carriage return character.

Format file:

```
HEADER ( "ONE TWO" )
//ROUND ( "6.1" )

<APPLICATION> " "
ROUND ( CCET ) " "
HEADER ( "THREE" )
COND ( (EQUALS(<APPLICATION>, "CCS"), CONCAT("00", CA)),
(TRUE,CONCAT("00", TN)) )
// end of line indicator:
\n \r
```

Input CDR file contents:

```
CCS|XYZ=123|CCET=0.2|TN=123123|CA=321321|ABC=333
ACS|XYZ=123|CCET=8.8|TN=123123|CA=321321|ABC=333
VPN|XYZ=123|CCET=-1.6|TN=123123|CA=321321|ABC=333
CCS|XYZ=123|CCET=BOB|TN=123123|CA=321321|ABC=333
```

Output file:

```
ONE TWO THREE
CCS 0 00321321
ACS 9 00123123
VPN -2 00123123
CCS 0 00321321
```

Further information

Because of the wide range of external EDR processing systems and site-specific requirements, it is not feasible in this document to describe all of the tasks which may be required to complete EDR integration.

For more information about this process, contact Level 1 support with details.

smsRecordStatistic

Purpose

This tool makes use of the SMS statistics subsystem, which in turn makes use of shared memory for communicating with the smsStatsDaemon. The smsStatsDaemon must be installed and running.

Location

The smsRecordStatistic process is located on the SLC in the **./IN/service_packages/SMS/bin** directory.

Configuration

smsRecordStatistic supports the following command-line options:

Usage:

```
smsRecordStatistic [application] [statistic] [value]
```

The available parameters are:

Parameter	Description
application	The name of the application for the statistic. (Optional)
statistic	The name of the statistic to record. (Optional)
value	Adds the given delta value to the statistic. (Optional.)

Output

The statistic named when running the script will be updated in the database.

smsStatsQuery

About smsStatsQuery

The smsStatsQuery utility enables you to directly query statistics generated on the Voucher and Wallet Server (VWS) and Service Logic Controller (SLC) before the statistics are replicated to the Service Management System (SMS). You use this utility for system monitoring.

Tip: You can view statistics that have been replicated to the SMS node by using the statistical viewing feature in the SMS user interface (UI). For more information, see *Service Management System User's Guide*.

The smsStatsQuery utility is located in the following directories:

- **/IN/service_packages/BE/bin** on the VWS nodes.
- **/IN/service_packages/SMS/bin** on the SLC nodes.

You can either supply a single query string as input to smsStatsQuery, or you can supply a text file containing a list of query strings as input.

Usage:

```
smsStatsQuery [options]  "stats_query"
smsStatsQuery [options] -f "queryfile"
```

Where:

- *options* is a space-separated list of optional parameters. The available options include options for the standard bc (binary calculator) Solaris utility, that is accessed by smsStatsQuery to apply calculations to the statistics results.

  The optional parameters and some typical bc options that you might want to set are listed in the *Optional Parameters Table* (on page 207) below.

  Note: You can display a full list of bc options by entering **`man bc`** at the UNIX prompt.
- *stats_query* is a string that identifies the statistics to query. To retrieve multiple statistics, specify a space-separated list of the statistics you want in the query string.

  You can also include a mathematical formula in the query to perform calculations on the retrieved statistics and return a single value.

  See *Specifying the Statistics to Query* (on page 206) for information about specifying the *stats_query* string.
- *queryfile* is the name of a text file that contains a list of *stats_query* strings.

Note: Specify either *stats_query* or *queryfile*, but not both.

Examples

In the following examples, the statistics to query are specified in a query string:

```
./smsStatsQuery "Acs_Service.elapsedTime"
./smsStatsQuery "Acs_Service.CALLS_INITIATED Acs_Service.ANNOUNCEMENTS_PLAYED"
```

In the following example, the statistics to query are specified in a text file:

```
./smsStatsQuery -f "queryFile.txt"
```

Note: The statistics on the VWS and SLC nodes are collected by smsStatsDaemon for replication to the SMS node. If you enter a query for a statistic that is not currently held by smsStatsDaemon then the smsStatsQuery utility returns an error. You can check which statistics are currently held by smsStatsDaemon by entering the following command:

```
./smsStatsQuery -l
```

For more information about smsStatsDaemon, see *smsStatsDaemon* (on page 160).

Specifying the Statistics to Query

To specify one or more statistics in the *stats_query* string, use the following syntax:

```
application.statistic[.detail] [application.statistic[.detail]]
```

Where:

- *application* is the name of the application or service that generated the statistic, such as Acs_Service.
- *statistic* is the name of the statistic to query, such as `elapsedTime`.
- *detail* (optional field) is the name of a detail field associated with the specified statistic. **Note:** Not all statistics have detail fields.

For example, the following queries retrieve statistics for the ACS service:

```
./smsStatsQuery "Acs_Service.elapsedTime"
./smsStatsQuery "Acs_Service.CALLS_INITIATED Acs_Service.ANNOUNCEMENTS_PLAYED"
```

To query a statistic that contains a space in any of its attribute names, you must use double square brackets, "[[" and "]]", to enclose the statistic specification in the *stats_query* string.

For example, the following queries include the "PrePaid Success" statistic in the *stats_query* string:

```
./smsStatsQuery "[[Ccs_Service.PrePaid Success]]"
./smsStatsQuery "Acs_Service.CALLS_INITIATED [[Ccs_Service.PrePaid Success]]
Acs_Service.ANNOUNCEMENTS_PLAYED"
```

To specify a formula in the *stats_query* string, use the following syntax:

```
[factor]statistic[[factor][statistic]]
```

Where:

- *factor* is a combination of a constant and an operator, or just an operator, that is applied to the statistic.
- *statistic* is a statistic specified as: *application.statistic.detail*.

For example:

```
./smsStatsQuery "10*Acs_Service.ANNOUNCEMENTS_PLAYED/Acs_Service.CALLS_INITIATED"
```

Note: You may not retrieve a list of statistics if you include a formula in the *stats_query* string.

Optional Parameters Table

This table describes the optional parameters for smsStatsQuery.

Parameter	Description
`-h`	Lists usage information for the smsStatsQuery utility.
`-l`	Lists the contents of statistics currently held by smsStatsDaemon.
`-t` *secs*	(bc option) Calculates the average rate a statistic is used based on two readings of the statistic, where the second reading is taken *secs* seconds after the first reading. The formula used to calculate the average rate a statistic is used is: (value of reading 2 - value of reading 1)/ *secs*.
`-p` *precision*	(bc option) Specifies the number of decimal places to display for the statistics value output.

startMerge

Purpose

This command initiates a master merge of an inferior master to a superior one. It can also be used to safely shut down an superior master by merging it with an inferior master.

Configuration

The startMerge supports the following command-line options:

Usage:

```
startMerge [-from nodenum] [-to nodenum]
```

The available parameters are:

Parameter	Default	Description
`-from` *nodenum*	none	Node number of the inferior master to merge from.
`-to` *nodenum*	none	Node number of the inferior master to merge to.

Failure

If startMerge fails, it will write an error to the syslog and exit.

Output

The startMerge writes error messages to the system messages file, and also writes additional output to **/IN/service_packages/SMS/tmp/merge.rep**.

Chapter 8

Reports

Overview

Introduction

This chapter explains SMS reporting functionality.

In this chapter

This chapter contains the following topics.

Reports Database Tables

Introduction

Report-generating functionality is available via the SMS Java screens, to provide for service management reports of data.

This topic describes how to create reports:

- Service reports, to be installed at the time of service installation
- General reports, installed subsequently

Database tables

There are three database tables which are specific to report generation:

- SMF_REPORT_SCRIPT contains one entry for each report script
- SMF_REPORT_PARAMETER contains one entry for each report parameter (may be none)
- SMF_REPORT_SCHEDULE contains one entry for each scheduled report instance. This table is not used for report installation, and is not covered in this document.

In addition, the following tables are used for controlling who has access to a report:

- SMF_APPLICATION
- SMF_APPLICATION_PART
- SMF_APPLICATION_ACCESS

These tables are reviewed here in regards to their role in report security. Security is handled by the standard SMS application part mechanism (see, example 3). Auditing is provided by the standard SMS audit mechanism, and should not need changing. The last change fields are the standard SMS last change fields, and are not listed in this table.

Report Scripts table

The report database table is called SMF_REPORT_SCRIPT. It contains the details of reports as shown below.

Field	Description
REPORT_ID	Unique identifier, primary key, generated by a counter.
APP_ID	Application ID, foreign key to SMF_APPLICATION(app_id)
PART_ID	Part ID for security, foreign key to SMF_APPLICATION_PART (part_id)
CATEGORY	Script category, identifies the subdirectory.
SCRIPT	Script name, identifies the **.sql** file or shell script to execute.
NAME	Name to list in the report directory.
DESCRIPTION	Help text.

Report parameter table

The parameter table is called SMF_REPORT_PARAMETER. It contains the details of report parameters as shown below.

Field	Description
REPORT_ID	The ID of the report this parameter belongs to.
PARAM_NUMBER	The position of the parameter in the list, for example 1st, 2nd.
NAME	The parameter name.
DESCRIPTION	Help text.
TYPE	The type – INT, STRING, DATE, and so on. (See table following for details)
DEFAULT_VALUE	Default value, optional.
VALID_VALUES	Valid comma separated values.
CONSTRAINT1	A constraint on the parameter (interpretation depends on TYPE).
CONSTRAINT2	A constraint on the parameter (interpretation depends on TYPE).

Installing a Report Script

Introduction

A script must be installed before it can be made available to the system. This process is described here, along with examples.

The main steps in the procedure detailed below are:

1. Choose an application ID, a category, and a report name.
2. Determine the parameters (if any) required by your script, and write the script.
3. Decide which application part your report will belong to.
4. Install the actual script on the SMS in the correct location.
5. Insert entries into the REPORT tables in the SMS database.

Procedure

Follow these steps to install a report script.

Step | **Action**

1 The Application ID must be an existing entry from SMF_APPLICATION. Common values are shown below. If you have additional services installed, additional choices may be available.

```
SQL> select app_id, application from smf_application;
    APP_ID APPLICATION
---------- --------------------
         1 SMS
         4 SYSTEM
         2 Acs_Service
```

The Category is an arbitrary name for a group of reports within one application. For example: "Customer", "Management", "Resource Usage".

The Name is a name for your report. Typically, this will be similar to the script name. For example, if your script is **monthly_usage.sql**, your report name could be "Monthly Usage".

2 Your script may take user parameters. The SMS report functions allow you to determine whether these are string, numeric, or values from a constrained list of parameters. Refer to the description of the SMF_REPORT_PARAMETER table to see the parameter types supported.

3 A report must one of the following:

- Have a **.sql** extension. In this case, it will be executed using sqlplus.
- Be executable by the smf_oper user.

In either case, the script will be passed (n + 1) command line parameters, where n is the number of user parameters defined in SMF_REPORT_PARAMETER. Command line parameter one will always be the absolute output file name allocated to this report.

Examples:

for a **.sql** file:

```
sqlplus script-name output-file [user-parameters]
```

for an executable file without a **.sql** extension

```
script-name output-file [user-parameters]
```

Exit status of report scripts are defined by the following:

0	= okay
>	= not okay. (Unix style)
<0	= undefined

Neither the smsReportsDaemon nor the smsReportScheduler is responsible for the clean up or reclaim of resources used by reports. This must be done explicitly by the application programmer.

The user may request cancelation of a script, in which case it will be sent a SIGTERM. Scripts should not ignore SIGTERM.

If the report spawns children, it should implement a SIGTERM handler to dispose the children, in case the user cancels a report.

4 For simplicity, an application part may be reused. Access to multiple reports may be controlled by one application part. You can even re-use access parts controlling existing installed screens.

You can list all existing defined application parts with the SQL command:

```
SQL> select app_id, part_id, part from smf_application_part;
```

If you choose to re-use application part 1030 (SMSReportScreens), all users who can access report screens will be able to run this report.

Step	Action
5	The script itself must be placed into **/IN/service_packages/SMS/input/***application-name***/***category***/***script-file*.
6	The script must now be made known to the SMS screens, and available for use to any SMS user who has access to the part ID, which owns your report script. This means: • Inserting an entry into SMF_REPORT_SCRIPT for your script, indicating the category and script filename. • Inserting one entry into SMF_REPORT_PARAMETER for each parameter in your report (if any). This indicates any constraints you wish enforced (for example, min/max values).

Report Script Worked Example

Introduction

A script must be installed before it can be made available to the system. This process is described here, along with examples.

The main steps are:

1 Choose a category and a name for your script.
2 Determine the parameters (if any) required by your script, and write the script.
3 Decide which application part your report will belong to.
4 Install the actual script on the SMS in the correct location.
5 Insert entries into the REPORT tables in the SMF database.

Example report script

Follow these steps to work through the example report script. The example script appears below description of each action in the procedure.

Step	Action
1	The details for this example are: Application: SMS (ID is 1) Category: “Errors” Name: “Program Errors” Script File: **program_errors.sql** Application Part: 1805 (new part) In this example, we are installing into the application SMS, which has the unique application ID of 1. Typically, you will install a service-specific report under the unique application ID that has been allocated to your service. We are free to choose the category; we have chosen the "Errors" category. We are free to choose a unique script name within this application and category; we have chosen **program_errors.sql**. We have chosen to create a new application part (1805) to control access to this report. An existing part may be used, for example: 1030.

Step	Action
2	The report takes three user parameters: Num Hours: Integer in range 1..999, default is 24 Program Prefix: String length 0..20 characters Category: One of FATAL, SERIOUS, WARNING, INFORMATIONAL **Note:** The script itself will take four parameters. The first parameter is the output file name, which is determined by the reports daemon and is handed to us. You *must* generate your output to that file, if you wish it to be seen by the user. The script below accepts three arguments, and shows the essential basic techniques of accepting input parameters, and spooling to the correct output file.

```
/*------------------------------------------------------------
 * File: program_errors.sql
 *
 * Updates:
 *
 * Parameters:      &1 Output file, determined by reports daemon
 *                  &2 Hours back
 *                  &3 Program prefix
 *                  &4 Severity
 *                  (FATAL,SERIOUS,WARNING,INFORMATIONAL)
 *

 * Copyright Notice:
 * (c)1998 This source code is owned and copyrighted by Oracle
 *-----------------------------------------------------------*/
-- #ident "@(#)$Id: telephony_errors.sql,v 1.4 1999/02/25 22:10:29 rhwang
Exp $"

-- so we won't print to stdout.
set termout off
set verify off

-- we are going to access the sms_program_errors table.
-- set the column titles.
column program 'Program' format a20
column error_code 'Error Code' format a16
column node_name 'Node Name' format a20
column severity 'Error Severity'

set linesize 80
set pagesize 2100

spool &1
-- now set the title at the top of the page.
ttitle center 'Recent Telephony Errors for Application &3 Severity &4'
skip 1 -
center 'PROGRAM TELEPHONY ERRORS' -
center =========================== skip 1 -
RIGHT 'PAGE:' FORMAT 999 SQL.PNO SKIP 2
```

Step	Action

```
break on program skip 1;
break on severity skip 1;

select program, severity, error_code, node_name, timestamp
from smf_program_errors
where (program like '%&3') and
      (severity like '%&4') and
      (timestamp < sysdate - &2/24)
order by program, severity;

spool off
quit
```

3 The user must now specify an application part for security purposes. If you have decided to re-use an existing part_id (for example: 1030), proceed to Example part 4.

a. The Application ID for this example is 1. This is the unique ID for SMS.

b. Application Part IDs must be in the range *App-ID* * 1000 + (0 … 999) so for SMS, this means 1000 .. 1999. In this example, it has been determined that the ID 1805 is available for use. This is a new ID, which will control access to this report (and possibly others placed in the same security domain).

c. Application Access IDs must be in the range <Part-ID> * 100 + (0..99). This is the part ID, so any Access ID within this range may be chosen. In the example, 180500 has been chosen.

As part of the installation for this script, run SQL to create this new Part ID.

Note: It is not necessary to create the SMF_APPLICATION for the SMS, since this is already created as part of the smsSms installation.

```
/*
 *  Create our application part. We can re-use this if we
 *  have multiple reports that we want to have all controlled
 *  by a single security identifier
 */
insert into smf_application_part (part_id, app_id, part, description)
  values (1805, 1, 'SMSErrorReports', 'Access SMS error reports
category');
insert into smf_application_access(access_id, part_id, rights_name,
description)
  values (180500, 1805, 'Access', 'Run reports');

/*
 *  We also add this to the 'SMS CreateDelete' user template,
 *  so that any user who is granted this template will get
 *  access to this report. We could add this to other
 *  templates too...
 */
var temp_id number;

EXEC select template_id into :temp_id from smf_template where
template='SMS CreateDelete';

insert into smf_template_access (template_id, access_id)
  values (:temp_id, 180500);

commit;
```

Step	Action
4	As part of the installation package, ensure that the file, **program_errors.sql** is installed into the correct destination location, for example, **/IN/service_packages/SMS/input/SMS/Errors/program_errors.sql**. The smf_oper user must have read access to this file. If this was a shell script or a binary program, it is necessary to ensure that the smf_oper also has execute access to this file.
5	The final task is to notify the SMS about the script, to make it visible. In this example, the report and the three user parameters to be collected are defined. **Note:** The screens constrain the content of the parameters to be passed to the script, but the interpretation of the parameters is of course up to the script itself.

```
/*
 * Add our script to the list of scripts.
 */
var report_ref number;

insert into smf_report_script
  (app_id, part_id, category, script, name, description)
values
  (1, 1805, 'Errors', 'program_errors.sql', 'Program Errors',
   'Dumps all the program errors for the specified program(s)');

exec select report_id into :report_ref -
from smf_report_script -
  where (app_id=1) -
  and (category = 'Errors') -
  and (name = 'Program Errors');

insert into smf_report_parameter (
  report_id, param_number, name, description, type,
  default_value, valid_values, constraint1, constraint2)
values
  (:report_ref, 1, 'Num Days', 'Number of hours to go back',
   'INT', '24', '', '1','999');

insert into smf_report_parameter (
  report_id, param_number, name, description, type,
  default_value, valid_values, constraint1, constraint2)
values
  (:report_ref, 2, 'User',
   'Leading string of program (0-20 characters)',
   'STRING', '', '', '0', '20');

insert into smf_report_parameter (
  report_id, param_number, name, description, type,
  default_value, valid_values, constraint1, constraint2)
values
  (:report_ref, 3, 'Category', 'Error Category (pulldown menu)',
   'STRING', 'FATAL', 'FATAL,SERIOUS,WARNING,INFORMATIONAL', '', '');

commit;
```

Database Auditing

Introduction

Changes to the data held in the SMF are tracked in the SMF_AUDIT table.

The listAudit.sh tool enables reports to be run on the changes tracked in SMF_AUDIT.

Purpose

listAudit.sh enables you to run queries against the audit data held in the SMF_AUDIT table. The results are processes in to a comma separated report.

Configuration

listAudit.sh accepts the following command line options.

Usage:

```
listAudit.sh usr/pwd [start_date] [end_date] [db_user] [table]
```

The available parameters are:

Parameter	Default	Description
`usr/pwd`		The user and password combination to be used to log into the SMF. (Required.)
start_date		The time and date the query will start reporting on. The format is yyyymmddhh24mmss. (Optional.)
end_date		The time and date the query will stop reporting on. The format is yyyymmddhh24mmss. (Optional.)
db_user		The userid for the database user which made the changes to the database. (Optional.)
table		The database table which was changed. (Optional.)

The square brackets indicate optional parameters, but if a parameter is missed out and a later one used the missed out parameters should be indicated by using "".

Failure

If listAudit.sh fails, the report will not be completed. Errors will be sent to stdout.

Output

listAudit.sh writes error messages to the system messages file, and produces reports to stdout.

Example: This text shows an audit report for changes to the SMF_USER table by the SU user on the 08 Mar 2005.

```
$ listAudit.sh smf/smf 20050308000000 20050308235959 SU SMF_USER
Connected.
SU,20050308225724,192168007165,SMF_USER,ADMIN_TRAINING1_EX2,Student Training,Student
1,0,,31,20050321010942,LANGUAGE=ENGLISH
,,,,ADMIN_TRAINING1_EX2,Student Training,Student 1,0,Locked for
testing,31,20050408000000,LANGUAGE=ENGLISH

SU,20050308225808,192168007165,SMF_USER,ADMIN_TRAINING1_EX1,Student Account,Student
1,0,,31,20050320205427,LANGUAGE=ENGLISH
,,,,ADMIN_TRAINING1_EX1,Student Account,Student 1,0,Locked for
training,31,20050408000000,LANGUAGE=ENGLISH
```

```
SU,20050308225828,192168007165,SMF_USER,ADMIN_TRAINING1_EX1,Student Account,Student
1,0,Locked for training,31,20050408000000,LANGUAGE=ENGLISH
,,,,ADMIN_TRAINING1_EX1,Student Account,Student
1,0,,31,20050408000000,LANGUAGE=ENGLISH

SU,20050308225838,192168007165,SMF_USER,ADMIN_TRAINING1_EX2,Student Training,Student
1,0,Locked for testing,31,20050408000000,LANGUAGE=ENGLISH
,,,,ADMIN_TRAINING1_EX2,Student Training,Student
1,0,,31,20050408000000,LANGUAGE=ENGLISH
```

Chapter 9

Troubleshooting

Overview

Introduction

This chapter explains the important processes on each of the server components in Convergent Charging Controller, and describes a number of example troubleshooting methods that can help aid the troubleshooting process before you raise a support ticket.

In this chapter

This chapter contains the following topics.

Common Troubleshooting Procedures

Introduction

Refer to *System Administrator's Guide* for troubleshooting procedures common to all Convergent Charging Controller components.

Checking current processes

You can check which processes are running using the standard UNIX command: ps. To find processes being run by Oracle software, you can grep for the string 'oper', which will display all processes being run by the application operator accounts (for example, acs_oper, ccs_oper and smf_oper).

Note: Some processes which are required for proper functioning may be run by other users, including root or the user which runs the webserver.

Example command: `ps -ef | grep oper`

For more information about the ps command, see the system documentation for the ps command.

You can also check how much of the processor a process is using by running the standard UNIX tool: top. If you have some baseline measurements, you will be able to compare it with the current load.

Example command: `top`

Tip: Some processes should only have one instance. If there are two or more instances, this may indicate a problem. For example, there will usually only be one timerIF running on each SLC.

For more information about which processes should be running on each node, check the Process List for each node in *Installation Guide*.

Restarting running processes using kill

Follow these steps to restart a running process.

Important: Restarting some processes can cause system instability or data loss. Some processes must be restarted using specific tools. Check the documentation for the process before restarting.

Step	Action
1	Find the Process ID for the process you want to restart. **Example command:** `ps -ef | grep smsAlarmRelay` **Note:** The second column of the results returned is the Process ID and the third column gives the Parent Process ID.
2	Kill the process using the kill command. **Example command:** `kill -TERM 123` **Result:** The process is terminated and will be restarted by the inittab process.

Checking configuration files

One of the significant areas where faults can occur and be remedied is in the configuration of processes. Configuration files can be edited by any standard text editor. A backup of the existing configuration file should always be taken before editing a configuration file.

For more information about the configuration files used in this application, see *Configuration User's Guide*.

For more information about the configuration file for a specific program or tool, see the section named after the binary in question.

Possible Problems

Introduction

This topic lists common problems and actions you can take to investigate or solve them. This list enables you to check for alarms based on the overall behavior you are experiencing.

SMS Java screens will not start

Follow these steps to resolve JavaClient problems.

Step	Action
1	Ensure that the HTTPD daemon (on the SMS) is running and that it is correctly configured.
2	If you are able to start the SMS screens, but unable to login: • Ensure that the **sms.jnlp** file is correctly configured. • Ensure that the SMS console is able to resolve host names into IP addresses.

Java help screen grayed out

This is caused by Java Runtime Environment (jre) running out of memory for the run time heap cache.

Under the default Java settings this may happen after 10 to 15 help screen accesses.

Follow these steps to extend the number of Help accesses.

Step	Action
1	Close the SMS screens.

Step	Action
2	From the Windows system, open the **Control Panel**.
3	Switch to **Classic View** to see the complete list of installed applications.
4	Double click **Java** icon to open java Control Panel.
5	Select the **Java** tab.
6	Click **View** in the **Java Applet Runtime Settings** panel.
7	Click the **Java Runtime Parameter** field. **Note:** This is the fourth field along, pop-up may require expanding to see this field.
8	Type `-Xms10M -Xmx512M` in the **Java Runtime Parameter** field. **Note:** If other parameters are there, add these to the end.
9	Click **OK**.
10	Click **Apply**.
11	Click **OK**.
12	Close the Control Panel.
13	Restart the browser and start the SMS screen.

Note: Using Xmx512M may cause issues with starting jre. If the browser jre cannot start up, try -Xmx180M.

Replication is failing

This table describes possible problems with replication.

Alarm	Reason	Remedy
`Cannot connect to Oracle - exiting`	There is a problem with the replication.config files in the system.	Use smsDumpRepConfig to check that the content of **replication.config** is correct. Generate a new **replication.config** file and check is it correctly copied to each machine. For more information, see *replication.config File* (on page 38).
`Could not make fifo f - exiting`	A connection is being dropped because the heartbeat settings on each end of a connection are different.	Check that the heartbeat settings for both ends of the connection are the same. The heartbeat settings are in **replication.def**, though they can be overridden at the command line for any process.

comparisonServer is failing

This table describes possible problems with comparisonServer.

Alarm	Reason	Remedy
	The **replication.config** file is not available to inetBootstrap, so smsCompareResync is not starting up.	Check that **replication.config** is in the correct directory and is readable by smf_oper.

Index Defragmentation

Description

The automatic defragmentation facility provided by SMS is intended to prevent fragmentation of the replication tables which frequently use insert, delete and update functions.

In order to enable this defragmentation facility, the script `fragmentation_install.sh` must first be installed. This will install the stored procedure `sms_defrag_rep_iot`, and schedule a job to run it every 10 minutes.

The following tables are affected:

- REP_ORA_EVENT
- REP_ORA_RENUMBERED

Before you begin

The process for installing the defragmentation script varies depending on the Oracle configuration available on the SMS. For most clustered environments, Oracle configuration is stored in the service parameter file (SPFILE), which permits configuration parameters to be modified at runtime. If this is the case, then the there is no need to manually alter the Oracle configuration.

However, if SPFILEs are not in use (that is the traditional PFILEs are used to manage Oracle configuration), then it is important to first modify the cache and block sizings in the **initSMF.ora** file. The cache size for the 32 KB block size should be set to 32 MB or another suitably large value.

Note: It is recommended that this activity is performed by an experienced DBA.

Enabling defragmentation

To enable the defragmentation facility, run the following script:

```
fragmentation_install.sh
```

This script is located in:

/IN/service_packages/SMS/db/defragmentation

Disabling defragmentation

To disable the defragmentation facility, run the uninstallation script as the oracle SMF user:

```
fragmentation_uninstall.sh
```

This script is located in:

/IN/service_packages/SMS/db/defragmentation

Oracle configuration restriction

While editing the parameter files, it must be noted that the following sets of parameters are mutually exclusive and cannot be used in combination with each other.

Example: You cannot use one or more of:

```
{db_cache_size,
db_recycle_cache_size,
db_keep_cache_size,
db_nk_cache_size (where n is one of 2,4,8,16,32),
db_cache_advice }
```

AND one or more of the following in your configuration:

```
{db_block_buffers
buffer_pool_keep
buffer_pool_recycle}
```

Chapter 10

Pre-installation

Overview

Introduction

This chapter explains the pre-installation configuration requirements of the application.

In this chapter

This chapter contains the following topics.

SMS Client Specifications

Specifications

This topic provides the specifications of SMS.

Network

The minimum requirements of network bandwidth for acceptable normal response times are as follows:

Number of Users	Minimum Requirements
1-5	512 KB
6-15	1 MB
16 +	LAN connection (at least 25% available resource of 10 MB)

Memory

The Convergent Charging Controller screens are written to optimize data interaction. As a result, it is necessary to cache data in such a way as to reduce redundant data retrieval. This means that heavy usage can lead to the requirement for a large amount of memory to be available on the client machine running the screens. The recommended memory installed on the client machine is 256MB minimum with 512 MB preferred, especially with machine running Windows XP.

This table shows the minimum client resources required.

RAM	CPU
256 MB	800 MHz

This table shows the recommended client resources required.

RAM	CPU
512 MB	1.2 GHz

Response Times

This table shows typical response time.

GUI Action	Response Time
Startup to Login dialog	30 seconds maximum
Login to SMS main screen	20 seconds maximum
SMS main screen to ACS	5 seconds maximum
ACS main screen to CPE	15 seconds maximum

Screen

Here is the required screen specification.

Pixel
800 x 600 pixel resolution

Preparing the System

Introduction

It is recommended that you check the kernel parameters on the system to ensure the system is optimally configured.

The following parameters are described in their respective technical guides. However, they are collated here for reference.

Note: Actual kernel parameters may be greater than those listed here.

Checking Kernel Parameters

Follow these steps to check the Kernel parameters for Solaris.

Step	Action
1	Log in as root.
2	Enter **`cat /etc/system`**
3	Check the parameters are set to at least the minimum values.
4	Change the parameters as required using the following command from **/etc/system**.

Parameters

This table shows all kernel parameters:

Parameter	Min Value	Hex Value	Description
`semmni`	1024		Number of semaphore identifiers.

Parameter	Min Value	Hex Value	Description
`semmsl`	1024		Maximum number of semaphores per unique ID.
`semmns`	14000		Maximum number of semaphores.
`shmmax`	4294967295	40000000	Maximum shared memory segment (bytes).
`shmmin`	1		Minimum shared memory segment (bytes).
`shmmni`	400		Number of shared memory identifiers.
`shmseg`	50		Number of shared memory segments allowed per process.
`semopm`	100		Maximum number of semaphore operations that can be executed per semop system call.
`semvmx`	65535		Maximum semaphore value.

Database Timezone and Backups

Setting Oracle timezone

To operate correctly, Oracle must be running on Greenwich mean time (GMT).

To ensure that Oracle is running on GMT, check that the following line is in the Oracle user's **.profile**:

```
TZ=GMT
export TZ
```

Oracle database domain

Check that the sqlnet.ora file does not override the oracle database domain specified in the initSMF*.ora file. Overriding will cause database creation failure with an inability to resolve the required database name in tnsnames.ora.

The **initSMF*.ora** files are located in the **/IN/service_packages/SMS/db/install/create/SMP/machine-profile** directory.

Each file should contain the following line:

```
db_domain=basms1p.SMF
```

The **sqlnet.ora** file should contain the following line:

```
NAMES.DEFAULT_DOMAIN = Oracle
```

The **sqlnet.ora** file will be in the **$ORACLE_HOME/network/admin/** directory.

Note: The specific **initSMF*.ora** file used in the installation is specific during the execution of the smsSms installation script.

SMF backups

The SMF can be backed up in two ways.

- Shut down the database periodically and backup all the database data files. This is simple but will disable provisioning and service side updates for the duration of the backup.
- Hot Backups:

  Archive logging should be enabled

  Archive logs and table spaces must be backed up and archive logs removed periodically. This procedure must be implemented by an individual with good knowledge of Oracle databases.

Archive logging

It is important to remember that if archive logging is enabled and the archive logs are not removed periodically then the disk will eventually fill up and the database will cease to function.

Starting Oracle Automatically on Reboot

Setting the initialization

In an operational environment, it is desirable that Oracle automatically start on reboot. This requires the creation of various scripts in the Unix initialization directories. A script is provided to simplify this task.

Before you begin

These tasks require that the "dbstart" script is in the default PATH for the "Oracle" user.

Procedure

Follow these steps to configure Oracle on Solaris.

Step	Action
1	Log on as root.
2	Enter `cd /IN/service_packages/SMS/install/init-sun`
3	Enter `sh ./oracle.sh` **Result:** The script will start. The output below shows the steps that are performed to configure automatic starting of Oracle on reboot.

```
Action: Configure Startup/Shutdown
Removing /etc/init.d/oracle
Removing /etc/rc.config.d/oracle
Removing /etc/rc2.d/S901oracle
Removing /etc/rc0.d/K011oracle
Copying oracle to /etc/init.d/oracle
/etc/rc.config.d exists
Creating /etc/rc.config.d/oracle
Creating link to /etc/rc2.d/S901oracle
Creating link to /etc/rc0.d/K011oracle
```

For Linux, see the discussion about automating shutdown and startup in *Oracle Database Administrator's Reference for Linux and UNIX-Based Operating Systems Guide*.

Chapter 11

About Installation and Removal

Overview

Introduction

This chapter provides information about the installed components for the Convergent Charging Controller application described in this guide. It also lists the files installed by the application that you can check for, to ensure that the application installed successfully.

In this Chapter

This chapter contains the following topics.

Installation and Removal Overview

Introduction

For information about the following requirements and tasks, see *Installation Guide*:

- Convergent Charging Controller system requirements
- Pre-installation tasks
- Installing and removing Convergent Charging Controller packages

SMS packages

An installation of Service Management System includes the following packages, on the:

- SMS:
  - smsSms
  - smsCluster (clustered)
  - efmSms
  - efmCluster (clustered)
- SLC:
  - smsScp
- VWS:
  - smsExtras

Raw Devices on Clustered SMS

Raw devices

SMS can allocate tablespace storage based on raw (without a file system) partitions. This enhances the performance of SMS on the SMS.

If you are using the raw devices option, you must create the raw partitions before installing the database using tools such as the system's format command.

The raw devices file (which you will be prompted to complete during the installation) must contain the full paths of the device files for the appropriate partitions.

The partitions must be at least as big as the required datafile sizings listed in the sizing file which will be used by the installation.

Example smf_devices.sh file

This is an example **smf_devices.sh** file.

```
#!/bin/sh
# The following file is the structure required for knowledge of
# raw device utilization and a few details pertaining to cluster
# creation. If clusters are not used then retaining the default
# values will be sufficient and not impact installation for raw
# device only.
#

# Details about the cluster
# How many instances in the cluster?
CLUSTER_INSTANCES=2
export CLUSTER_INSTANCES

# For each instance in the cluster we need to know the node name
# to install into the service configuration
NODE_1=smp1
NODE_2=smp2
export NODE_1 NODE_2

# These are the generic RAW DEVICE requirements for the cluster
# NOTE:// ENSURE ALL THESE DEVICES ARE READ WRITEABLE BY THE
# ORACLE USER OTHERWISE INSTALLATION WILL FAIL
# System tablespace
SYSTEM_TABLESPACE=/dev/did/rdsk/d10s5
export SYSTEM_TABLESPACE

# USERS tablespace
USERS_TABLESPACE=/dev/did/rdsk/d10s6
export USERS_TABLESPACE

# Temporary tablespace
TEMP_DATAFILE_1=/dev/did/rdsk/d10s2
TEMP_DATAFILE_2=/dev/did/rdsk/d10s3
TEMP_DATAFILE_3=/dev/did/rdsk/d10s4
TEMP_DATAFILE_4=/dev/did/rdsk/d11s0
TEMP_DATAFILE_5=/dev/did/rdsk/d11s1
TEMP_DATAFILE_6=/dev/did/rdsk/d11s2
TEMP_DATAFILE_7=/dev/did/rdsk/d11s3
TEMP_DATAFILE_8=/dev/did/rdsk/d11s4
export TEMP_DATAFILE_1 TEMP_DATAFILE_2 TEMP_DATAFILE_3
export TEMP_DATAFILE_4 TEMP_DATAFILE_5 TEMP_DATAFILE_6
export TEMP_DATAFILE_7 TEMP_DATAFILE_8
```

```
# Tools tablespace
TOOLS_TABLESPACE=/dev/did/rdsk/d10s7
export TOOLS_TABLESPACE

# 3 control file devices
CONTROL_FILE_1=/dev/did/rdsk/d12s3
CONTROL_FILE_2=/dev/did/rdsk/d12s4
CONTROL_FILE_3=/dev/did/rdsk/d12s5
export CONTROL_FILE_1 CONTROL_FILE_2 CONTROL_FILE_3

# Service Configuration Device
SRVM=/dev/did/rdsk/d11s5
export SRVM

# Now the UNDO tables. There will be 1 UNDO tablespace per instance in the
# cluster, having 5 datafiles per tablespace
# Standard to use here is UNDOTBS${NODEID}_DATAFILE_X, so UNDOTBS1
# is the UNDO space for NODE_1
# If clusters are not in use and this is raw device then UNDOTBS1
# only needs populating.
UNDOTBS1_DATAFILE_1=/dev/did/rdsk/d9s0
UNDOTBS1_DATAFILE_2=/dev/did/rdsk/d9s1
UNDOTBS1_DATAFILE_3=/dev/did/rdsk/d9s2
UNDOTBS1_DATAFILE_4=/dev/did/rdsk/d9s3
UNDOTBS1_DATAFILE_5=/dev/did/rdsk/d9s4
export UNDOTBS1_DATAFILE_1 UNDOTBS1_DATAFILE_2 UNDOTBS1_DATAFILE_3
export UNDOTBS1_DATAFILE_4 UNDOTBS1_DATAFILE_5

# We require one of the following UNDOTBS sections PER cluster instance
# The ** REQUIRED ** format is UNDOTBSX_DATAFILE_Y= where X is the instance
# ID of the node defined in NODE_X and Y is the log file number
UNDOTBS2_DATAFILE_1=/dev/did/rdsk/d9s5
UNDOTBS2_DATAFILE_2=/dev/did/rdsk/d9s6
UNDOTBS2_DATAFILE_3=/dev/did/rdsk/d9s7
UNDOTBS2_DATAFILE_4=/dev/did/rdsk/d10s0
UNDOTBS2_DATAFILE_5=/dev/did/rdsk/d10s1
export UNDOTBS2_DATAFILE_1 UNDOTBS2_DATAFILE_2 UNDOTBS2_DATAFILE_3
export UNDOTBS2_DATAFILE_4 UNDOTBS2_DATAFILE_5

# And the redo logs. The sizing is for $REDO_LOGS_PER_NODE redo logs per
# node in the cluster, so this section requires $CLUSTER_INSTANCES *
# $REDO_LOGS_PER_NODE to be complete. Naming standard is
# redo$NODEID_X, for example REDO1_1, REDO1_2 ...
REDO_LOGS_PER_NODE=16
export REDO_LOGS_PER_NODE

REDO1_1=/dev/did/rdsk/d5s0
REDO1_2=/dev/did/rdsk/d5s1
REDO1_3=/dev/did/rdsk/d5s2
REDO1_4=/dev/did/rdsk/d5s3
REDO1_5=/dev/did/rdsk/d5s4
REDO1_6=/dev/did/rdsk/d5s5
REDO1_7=/dev/did/rdsk/d5s6
REDO1_8=/dev/did/rdsk/d5s7
REDO1_9=/dev/did/rdsk/d6s0
REDO1_10=/dev/did/rdsk/d6s1
REDO1_11=/dev/did/rdsk/d6s2
REDO1_12=/dev/did/rdsk/d6s3
REDO1_13=/dev/did/rdsk/d6s4
REDO1_14=/dev/did/rdsk/d6s5
REDO1_15=/dev/did/rdsk/d6s6
REDO1_16=/dev/did/rdsk/d6s7
export REDO1_1 REDO1_2 REDO1_3 REDO1_4
export REDO1_5 REDO1_6 REDO1_7 REDO1_8
export REDO1_9 REDO1_10 REDO1_11 REDO1_12
export REDO1_13 REDO1_14 REDO1_15 REDO1_16


# As with the UNDOTBS we require a set of redo logs per nodal instance in
# the cluster. The format ** REQUIRED ** is REDOX_Y= where X is the instance
# ID of the node defined in NODE_X and Y is the log file number
REDO2_1=/dev/did/rdsk/d7s0
REDO2_2=/dev/did/rdsk/d7s1
REDO2_3=/dev/did/rdsk/d7s2
REDO2_4=/dev/did/rdsk/d7s3
REDO2_5=/dev/did/rdsk/d7s4
REDO2_6=/dev/did/rdsk/d7s5
REDO2_7=/dev/did/rdsk/d7s6
REDO2_8=/dev/did/rdsk/d7s7
REDO2_9=/dev/did/rdsk/d8s0
REDO2_10=/dev/did/rdsk/d8s1
REDO2_11=/dev/did/rdsk/d8s2
REDO2_12=/dev/did/rdsk/d8s3
REDO2_13=/dev/did/rdsk/d8s4
REDO2_14=/dev/did/rdsk/d8s5
REDO2_15=/dev/did/rdsk/d8s6
REDO2_16=/dev/did/rdsk/d8s7
export REDO2_1 REDO2_2 REDO2_3 REDO2_4
export REDO2_5 REDO2_6 REDO2_7 REDO2_8
export REDO2_9 REDO2_10 REDO2_11 REDO2_12
export REDO2_13 REDO2_14 REDO2_15 REDO2_16

# SMS Specific
SMF_DATA_DATAFILE=/dev/did/rdsk/d11s6
SMF_INDEX_DATAFILE=/dev/did/rdsk/d11s7
SMF_LOGS_DATAFILE_1=/dev/did/rdsk/d12s0
SMF_LOGS_DATAFILE_2=/dev/did/rdsk/d12s1
SMF_LOGS_INDEX_DATAFILE=/dev/did/rdsk/d12s2
export SMF_DATA_DATAFILE SMF_INDEX_DATAFILE
export SMF_LOGS_DATAFILE_1 SMF_LOGS_DATAFILE_2
export SMF_LOGS_INDEX_DATAFILE
```

Setting up ssh keys

Introduction

Some of the processes in SMS use ssh and scp to transfer data around the network. Consequently, ssh keys and permissions need to be set up on the relevant machines.

Procedure

Follow these steps to generate an automatic ssh access to a replication node.

Step	Action
1	Log into the SMS host as `smf_oper`.
2	Enter `ssh smf@`*`host`* where *`host`* stands for the replicated node host, for example, XXSCP1

Step	Action

3	Run the `ssh-keygen` package.
	Example command: `ssh-keygen`
	Result: The script will display the following prompts one at a time:
	`Enter file in which to save the key(/IN/service_packages/SMS/.ssh/id_rsa):`
	`Generating public/private rsa key pair.`
	`Enter passphrase(empty for no passphrase):`
	`Enter same passphrase again:`

4	Press **Enter** to continue. **Result:** The ssh public key will be generated and saved in **.ssh/id_rsa.pub**.
5	Log into the replicated node host, for example, XXSCP1, as `smf_oper`.
6	Append the content of the public key to authorized keys. **Example command:** `cat .ssh/id_rsa.pub >> .ssh/authorized_keys`
7	Test the ssh access on the replicated node. **Example command:** `ssh smf_oper@`*`host`*

Checking the Installation

Introduction

Refer to these checking procedures to ensure that SMS has installed correctly.

The end of the smsSms installation process (both unclustered and clustered) specifies a script designed to check the installation just performed. They must be run from the command line.

Check unclustered SMS procedure

Follow these steps to ensure SMS has been installed on an unclustered SMS machine correctly.

Step	Action
1	Log in to SMS machine as root.
2	Check the following directory structure exists with subdirectories: • **/IN/service_packages/SMS** • **/IN/html**
3	Check both directories contain subdirectories and that all are owned by: smf_oper user (group oracle)
4	Log into the system as smf_oper. **Note:** This step is to check that the smf_oper user is valid.
5	Check that the permissions for smf_oper's .ssh directory are: `dwrx------` **Note:** These permissions are required for the ssh keys to work correctly.

Step	Action
6	Type `sqlplus /` No password is required. **Note:** This step is to check that the smf_oper user has valid access to the database.

7	Check the smsSms.install.log has finished with the lines
	`* To validate this install, run:`
	`* smsInstallCheck -u smf/smf -f /IN/service_packages/SMS/etc/smsSms.check`
	Note: This log will contain 375 (at Version 3.0.0.15) spurious Errors related to drop table/ view and alter table commands where no object exits. This is caused by underlying Oracle routines that always drop before creation and can be safely ignored.

8	Ensure that the required SMS tables have been added to the database. To do this, execute the smsInstallCheck program with the following parameters: `~smf_oper/bin/smsInstallCheck -u smf/`*`smf_password`*` -f /IN/service_packages/SMS/etc/smsSms.check` **Result:** You should see the following output:

```
CHECKING: This program is run with -u SMF/<password>.
... OK

CHECKING: User smf_oper program smsMaster has 2 instances.
... OK

CHECKING: User smf_oper program smsReportsDaemon has 1 instances.
... OK

CHECKING: User smf_oper program smsReportScheduler has 1 instances.
... OK

CHECKING: User smf_oper program smsAlarmRelay has 1 instances.
... OK

CHECKING: User smf_oper program smsNamingServer has 1 instances.
... OK

CHECKING: User smf_oper program smsAlarmDaemon has 1 instances.
... OK

CHECKING: User smf_oper program smsTaskAgent has 1 instances.
... OK

CHECKING: Logged OraUser (SMF) has 23 tables like 'SMF_%'.
... OK


CHECKING: Logged OraUser (SMF) has 10 tables like 'REP_%'.
... OK

CHECKING: Logged OraUser (SMF) has 1 tables like 'IORS'.
... OK

CHECKING: Logged OraUser (SMF) has 62 valid triggers like 'SMF_%'.
... OK

CHECKING: Logged OraUser (SMF) has 22 valid triggers like 'REP_%'.
... OK

CHECKING: Logged OraUser (SMF) has 2 valid triggers like 'IORS_%'.
... OK

CHECKING: Logged OraUser (SMF) has 1 valid triggers like 'SSD_%'.
... OK

CHECKING: Logged OraUser (SMF) has 1 valid triggers like 'SSR %'.
```

Step	Action

	`... OK`
	`Dec 29 21:30:34 cmnError(29188) NOTICE: smsInstallCheck: Passed all 16 tests.`

9	Check the entries of the **/etc/inittab** file. Inittab entries reserved for SMS on SMS: a. `sms7   /IN/service_packages/SMS/bin/smsMasterStartup.sh` b. (runs smsMaster) c. `sms9   /IN/service_packages/SMS/bin/smsMergeDaemonStartup.sh` d. (runs smsMergeDaemon) e. `sms5 /IN/service_packages/SMS/bin/smsAlarmDaemonSmsStartup.sh` f. (runs smsAlarmDaemon) g. `sms1   /IN/service_packages/SMS/bin/smsAlarmRelayStartup.sh` h. (runs smsAlarmRelay) i. `sms6 /IN/service_packages/SMS/bin/smsStatsThresholdStartup.sh` j. (runs smsStatsThreshold) k. `sms4 /IN/service_packages/SMS/bin/smsReportSchedulerStartup.sh` l. (runs smsReportScheduler) m. `sms3 /IN/service_packages/SMS/bin/smsReportsDaemonStartup.sh` n. (runs smsReportsDaemon) o. `sms2 /IN/service_packages/SMS/bin/smsNamingServerStartup.sh` p. (runs smsNamingServer) q. `sms8   /IN/service_packages/SMS/bin/smsTaskAgentStartup.sh` r. (runs smsTaskAgent)
10	Check that the processes listed in the process lists are running on the relevant machine. For a list of the processes which should be running, see Process list - unclustered SMP.

Check clustered SMS procedure

Follow these steps to ensure SMS has been installed on a clustered SMS machine correctly.

Step	Action
1	Log in to SMS machine as root.
2	Check the following directory structure exists with subdirectories: • **/IN/service_packages/SMS** • **/IN/html**
3	Check both directories contain subdirectories and that all are owned by: smf_oper user (group oracle)
4	Log into the system as smf_oper. **Note:** This step is to check that the smf_oper user is valid.
5	Check that the permissions for smf_oper's **.ssh** directory are: `dwrx------`

Step	Action
	Note: These permissions are required for the ssh keys to work correctly.
6	Type `sqlplus /` No password is required. **Note:** This step is to check that the smf_oper user has valid access to the database.

7	Check the smsSms.install.log has finished with the lines
	`* To validate this install, run:`
	`* smsInstallCheck -u smf/smf -f /IN/service_packages/SMS/etc/smsSms.check`
	Note: This log will contain 375 (at Version 3.0.0.15) spurious Errors related to drop table/ view and alter table commands where no object exits. This is caused by underlying Oracle routines that always drop before creation and can be safely ignored.

8	Ensure that the required SMS tables have been added to the database. To do this, execute the smsInstallCheck program with the following parameters: `~smf_oper/bin/smsInstallCheck -u smf/`*`smf_password`*` -f /IN/service_packages/SMS/etc/smsSms.check` **Result:** You should see the following output:

```
CHECKING: This program is run with -u SMF/password.
... OK

CHECKING: User smf_oper program smsMaster has 2 instances.
... OK

CHECKING: User smf_oper program smsReportsDaemon has 1 instances.
... OK

CHECKING: User smf_oper program smsReportScheduler has 1 instances.
... OK

CHECKING: User smf_oper program smsAlarmRelay has 1 instances.
... OK

CHECKING: User smf_oper program smsNamingServer has 1 instances.
... OK

CHECKING: User smf_oper program smsAlarmDaemon has 1 instances.
... OK

CHECKING: User smf_oper program smsTaskAgent has 1 instances.
... OK

CHECKING: Logged OraUser (SMF) has 23 tables like 'SMF_%'.
... OK

CHECKING: Logged OraUser (SMF) has 10 tables like 'REP_%'.
... OK


CHECKING: Logged OraUser (SMF) has 1 tables like 'IORS'.
... OK

CHECKING: Logged OraUser (SMF) has 62 valid triggers like 'SMF_%'.
... OK

CHECKING: Logged OraUser (SMF) has 22 valid triggers like 'REP_%'.
... OK

CHECKING: Logged OraUser (SMF) has 2 valid triggers like 'IORS_%'.
... OK

CHECKING: Logged OraUser (SMF) has 1 valid triggers like 'SSD_%'.
```

Step	Action

	`... OK`
	
	`CHECKING: Logged OraUser (SMF) has 1 valid triggers like 'SSR_%'.`
	`... OK`
	`Dec 29 21:30:34 cmnError(29188) NOTICE: smsInstallCheck: Passed all 16 tests.`
	
	Note: The smsInstallCheck program will report missing processes. This is because the cluster resources have not been set up yet by smsCluster.

9	Check the entries of the **/etc/inittab** file. Inittab entries reserved for SMS on SMS: `sms7   /IN/service_packages/SMS/bin/smsMasterStartup.sh` (runs smsMaster)
10	Ensure the following shell scripts are configured to be run by the clustering software: • `/IN/service_packages/SMS/bin/smsAlarmDaemonSmsCluster.sh` (runs smsAlarmDaemon) • `/IN/service_packages/SMS/bin/smsAlarmRelayCluster.sh` (runs smsAlarmRelay) • `/IN/service_packages/SMS/bin/smsStatsThresholdCluster.sh` (runs smsStatsThreshold) • `/IN/service_packages/SMS/bin/smsReportSchedulerCluster.sh` (runs smsReportScheduler) • `/IN/service_packages/SMS/bin/smsReportsDaemonCluster.sh` (runs smsReportsDaemon) • `/IN/service_packages/SMS/bin/smsNamingServerCluster.sh` (runs smsNamingServer) • `/IN/service_packages/SMS/bin/smsTaskAgentCluster.sh` (runs smsTaskAgent)
11	Check that the processes listed in the process lists are running on the relevant machine.

Check SLC procedure

Follow these steps to ensure SMS has been installed on the SLC machine correctly.

Step	Action
1	Log in to SLC machine as root.
2	Check the following directory structure exists with subdirectories: **/IN/service_packages/SLEE**.
3	Check both directories contain subdirectories and that all are owned by: smf_oper user (group oracle)
4	Log into the system as smf_oper. **Note:** This step is to check that the smf_oper user is valid.
5	Check that the permissions for smf_oper's .ssh directory are: `dwrx------` **Note:** These permissions are required for the ssh keys to work correctly.

Step	Action
6	Type `sqlplus /` No password is required. **Note:** This step is to check that the smf_oper user has valid access to the database.
7	Ensure that the required ACS triggers have been added to the database for the ACS_ADMIN oracle user.
8	Check the entries of the **/etc/inittab** file. Inittab entries reserved for SMS on SLC: • `scp1   /IN/service_packages/SMS/bin/cmnPushFilesStartup.sh` (runs cmnPushFiles) • `scp2   /IN/service_packages/SMS/bin/infMasterStartup.sh` (runs infMaster) • `scp3   /IN/service_packages/SMS/bin/smsStatsDaemonStartup.sh` (runs smsStatsDaemon) • `scp4 /IN/service_packages/SMS/bin/smsAlarmDaemonScpStartup.sh` (runs smsAlarmDaemon) • `scp5   /IN/service_packages/SMS/bin/updateLoaderStartup.sh` (runs updateLoader)
9	Check that the processes listed in the process lists are running on the relevant machine

Check other machines procedure

Follow these steps to ensure SMS has been installed correctly on machines other than SMSs or SLCs.

Step	Action
1	Log in to the machine as root.
2	Check the following directory structure exists with subdirectories: **/IN/service_packages/SMS**
3	Check both directories contain subdirectories and that all are owned by: smf_oper user (group oracle)
4	Log into the system as smf_oper. **Note:** This step is to check that the smf_oper user is valid.
5	Check that the permissions for smf_oper's .ssh directory are: `dwrx------` **Note:** These permissions are required for the ssh keys to work correctly.
6	If a database has been installed on the machine, and SMS statistics has been configured to use the database, type `sqlplus /` No password is required. **Note:** This step is to check that the smf_oper user has valid access to the database.
7	Ensure that the required SMS tables have been added to the database for the SMF oracle user.
8	Check the entries of the **/etc/inittab** file:

Step	Action
	Inittab entries reserved for SMS on SLC: **1** `ext8   /IN/service_packages/SMS/bin/smsStatsDaemonStartup.sh` (runs smsStatsDaemon) **2** `ext9   /IN/service_packages/SMS/bin/smsAlarmDaemonStartup.sh` (runs smsAlarmDaemon)
9	Check that the processes listed in the process lists are running on the relevant machine. For a list of the processes which should be running, see *Process list - other machines* (on page 240).

Process list

If the application is running correctly, the following processes should be running on each SMS, started from the inittab:

- smsMaster
- smsMergeDaemon
- smsAlarmDaemon
- smsAlarmManager
- smsAlarmRelay
- smsStatsThreshold
- smsReportScheduler
- smsReportsDaemon
- smsNamingServer
- smsTaskAgent
- smsTrifDaemon

Process list - clustered SMS

If the application is running correctly, the following processes should be running on each SMS.

- smsMaster, started from the inittab.
- The following are started by the clustering software.
  - smsAlarmDaemon
  - smsAlarmManager
  - smsAlarmRelay
  - smsStatsThreshold
  - smsReportScheduler
  - smsReportsDaemon
  - smsNamingServer
  - smsTaskAgent
  - smsTrigDaemon

Process list - SLC

If the application is running correctly, the following processes should be running on each SLC, started from the inittab:

- infMaster
- updateLoader
- smsAlarmDaemon

- smsStatsDaemon
- cmnPushFiles

Process list - other machines

If the application is running correctly, the following processes should be running on each platform, started from the inittab:

- If alarms use replication, smsAlarmDaemon
- If statistics use replication, smsStatsDaemon

Check the SMS Administration Screens

Check that the SMS administration screens are working correctly. Use an internet browser to open the following web page:

```
http://smshostname
```

Where:

smshostname is the hostname of an SMS in the IN.

Launch the application using the Webstart link. For more information about using the SMS Java administration screens, see *Service Management System User's Guide*.

Check alarm replication

Follow these steps to check that alarm replication is functioning correctly.

Step	Action
1	Open the SMS Java administration screen.
2	Open the **Operator Functions >** Node Management screen.
3	Click **Create**. **Result:** This will create a **replication.config** file and distribute a copy to all the machines in the IN.
4	Check that the file exists on all the machines in the IN.
5	On each node in the IN, use smsLogTest to generate an error. For more information about smsLogTest, see *smsLogTest* (on page 196).
6	Open the **Operator Functions >** Alarm Management screen.
7	Check that the alarm has replicated from each node into the SMF_ALARM_MESSAGE table in the SMF.

Enabling index defragmentation

Once the installation process is completed, it is advisable to enable the index defragmentation facility, although it is not strictly necessary.

Note: This facility has a dependency on specific Oracle configuration settings which relate to the nature of deployment. For more information how to install the defragmentation script, see *Index defragmentation*.

Glossary of Terms

ACS

Advanced Control Services configuration platform.

ANI

Automatic Number Identification - Term used in the USA by long-distance carriers for CLI.

API

Application Programming Interface

ASN.1

Abstract Syntax Notation One - a formal notation used for describing data transmitted by telecommunications protocols. ASN.1 is a joint ISO/IEC and ITU-T standard.

BFT

Billing Failure Treatment - the process that is applied if the system has lost all connections to a billing engine. It allows for limited continuation of call processing functions, if configured.

CC

Country Code. Prefix identifying the country for a numeric international address.

CCS

1) Charging Control Services (or Prepaid Charging) component.

2) Common Channel Signalling. A signalling system used in telephone networks that separates signalling information from user data.

CDR

Call Data Record

Note: The industry standard for CDR is EDR (Event Detail Record).

CID

Call Instance Data

CLI

Calling Line Identification - the telephone number of the caller. Also referred to as ANI.

Connection

Transport level link between two peers, providing for multiple sessions.

Convergent

Also “convergent billing”. Describes the scenario where post-paid and pre-paid calls are handed by the same service platform and the same billing system. Under strict converged billing, post-paid subscribers are essentially treated as “limited credit pre-paid”.

CORBA

Common Object Request Broker Architecture. It is a framework that provides interoperability between objects built in different programming languages, running on different physical machines perhaps on different networks. It specifies an Interface Definition Language, and API that allows client / server interaction with the ORB.

CPE

Control Plan Editor (previously Call Plan Editor) - software used to define the logic and data associated with a call -for example, "if the subscriber calls 0800 *nnnnnn* from a phone at location *xxx* then put the call through to *bb bbb bbbb*".

CPU

Central Processing Unit

cron

Unix utility for scheduling tasks.

crontab

File used by cron.

DB

Database

DTMF

Dual Tone Multi-Frequency - system used by touch tone telephones where one high and one low frequency, or tone, is assigned to each touch tone button on the phone.

GUI

Graphical User Interface

HTML

HyperText Markup Language, a small application of SGML used on the World Wide Web.

It defines a very simple class of report-style documents, with section headings, paragraphs, lists, tables, and illustrations, with a few informational and presentational items, and some hypertext and multimedia.

HTTP

Hypertext Transport Protocol is the standard protocol for the carriage of data around the Internet.

IN

Intelligent Network

initab

The initab holds the start up and configuration information for most of the processes used in SMS. The default location is /etc/initab.

IOR

Inter-operable Object Reference. A reference that is used in the CORBA world that clients can use to send their requests to a particular process executing on a particular machine. Every CORBA based server has an IOR that uniquely identifies it within a distributed computing platform. IOR consists of information such as the IP address of the machine on which the process is executing, or the port number to which it is listening. This IOR is usually exported/sent to some form of central registry when the process is started up. Clients can then retrieve this information, that is, IORs, from the central registry if they want to send a request to a server.

IP

1) Internet Protocol

2) Intelligent Peripheral - This is a node in an Intelligent Network containing a Specialized Resource Function (SRF).

IP address

Internet Protocol Address - network address of a card on a computer.

ITU

International Telecommunication Union

MID

Measurement ID - used in Number Portability, counts the occurrences of an error.

ORB

Object Request Broker. Within an Object based communication system, an ORB keeps track of the actual addresses of all defined objects and thus is used to route traffic to the correct destination. The CORBA defines the ORB in a series of standards enabling different platforms to share common information.

PC

Point Code. The Point Code is the address of a switching point.

PI

Provisioning Interface - used for bulk database updates/configuration instead of GUI based configuration.

PIN

Personal Identification Number

SCP

Service Control Point. Also known as SLC.

SGML

Standard Generalized Markup Language. The international standard for defining descriptions of the structure of different types of electronic document.

SLC

Service Logic Controller (formerly UAS).

SLEE

Service Logic Execution Environment

SLPI

Service Logic Program Instance

SMP

Service Management Platform (also referred to as SMS).

SMS

Depending on context, can be:

- Service Management System hardware platform
- Short Message Service
- Service Management System platform
- Convergent Charging Controller Service Management System application

SN

Service Number

SNMP

Simple Network Management Protocol. Usually responsible for notifying faults on a network.

SQL

Structured Query Language - a database query language.

SRF

Specialized Resource Function – This is a node on an IN which can connect to both the SSP and the SLC and delivers additional special resources into the call, mostly related to voice data, for example play voice announcements or collect DTMF tones from the user. Can be present on an SSP or an Intelligent Peripheral (IP).

SSF

Sub Service Field.

SSL

Secure Sockets Layer protocol

SSP

Service Switching Point

SUA

Signalling Connection Control Part User Adaptation Layer

System Administrator

The person(s) responsible for the overall set-up and maintenance of the IN.

TCAP

Transaction Capabilities Application Part – layer in protocol stack, message protocol.

TCP

Transmission Control Protocol. This is a reliable octet streaming protocol used by the majority of applications on the Internet. It provides a connection-oriented, full-duplex, point to point service between hosts.

TLS

Transport Layer Security. Cryptographic protocol used to provide secure communications. Evolved from SSL.

URL

Uniform Resource Locator. A standard way of specifying the location of an object, typically a web page, on the Internet.

VWS

Oracle Voucher and Wallet Server (formerly UBE).

XML

eXtensible Markup Language. It is designed to improve the functionality of the Web by providing more flexible and adaptable information identification.

It is called extensible because it is not a fixed format like HTML. XML is a `metalanguage' — a language for describing other languages—which lets you design your own customized markup languages for limitless different types of documents. XML can do this because it's written in SGML.

Index