Oracle® Communications Network Charging and Control

Service Logic Execution Environment Technical Guide

Release 15.2

January 2026

Copyright

Contents

About This Document

Scope

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

Audience

This guide was written primarily for system administrators and persons installing, configuring and administering the SLEE 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

There are no documents related to this document.

Document Conventions

Typographical Conventions

The following terms and typographical conventions are used in the Oracle Communications Network Charging and Control (NCC) 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.

Terminology

This topic explains any terminology specific to this manual.

Term	Definition
ACS Customer	ACS customers are set up in the ACS Customer screen. They configure systems to provide services to subscribers.
Service Provider	CCS service providers are the same as ACS customers and are set up in the ACS Customer screen. There is additional service provider configuration provided in CCS.
Customer	Customers in CCS refer to the customers configured on the Subscriber Management screen.
Subscriber	A subscriber account is set up for each MSISDN which uses services provided by the service provider.

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 Network Charging and Control (NCC) network or service implications of the product.

In this Chapter

This chapter contains the following topics.

Introduction to the Service Logic Execution Environment

Introduction

The Service Logic Execution Environment (SLEE) provides a common execution environment for existing Oracle NCC products, including:

- Advanced Control Services (ACS)
- Charging Control Services (CCS)
- Voucher and Wallet Server (VWS)
- Messaging Manager (MM)

It provides mechanisms for multiple interfaces to communicate events with the call, therefore simplifying the service logic interfaces.

Functionality overview

The SLEE provides the following functionality:

- SLEE process monitoring/restart
- Simultaneous management of multiple different service logic applications.

Functionality for hosted services

The main functions of the SLEE are to provide hosted services with the following functionality:

- Event handling/call matching
- Event scheduling
- Call thread and context data control

- Service logic application management
- Interface management.

Service logic support

The SLEE implements the common components of service logic within a single environment.

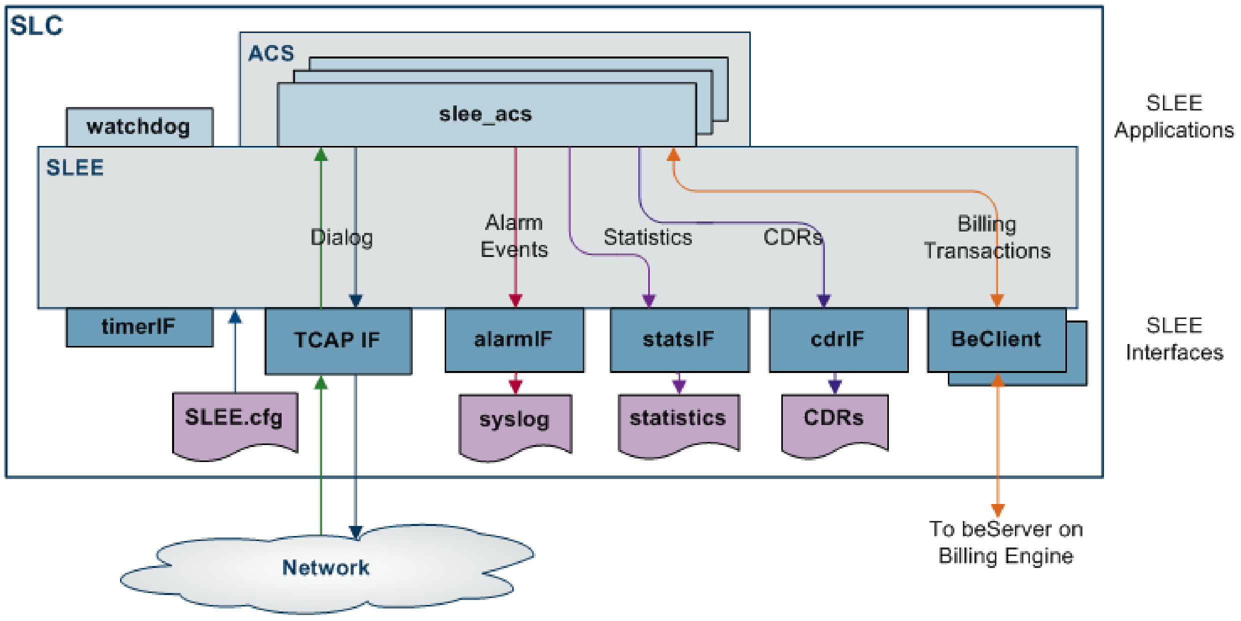
It provides the following to service logic interfaces:

- A well-defined, open interface for the handling of call control threads, call context data and application management
- Efficient flexible mechanisms for multiple interfaces to communicate events with the call.

The SLEE maintains integrity and ensures high performance when managing multiple messages from multiple underlying networks to multiple applications.

Example setup

This figure shows a typical SLEE setup.



Main Components of the SLEE

Introduction

The NCC SLEE uses the following components.

Component	Description	Further Information
SLEE.cfg	This file holds the main configuration for the SLEE and startup.	*Configuring the SLEE* (on page 10)
SLEE shared memory	Shared memory used by SLEE applications and interfaces.	
SLEE API	Application Programming Interface used by SLEE applications and interfaces.	
watchdog	The SLEE watchdog monitors SLEE applications and interfaces, restarting them if necessary.	*watchdog* (on page 20)

Component	Description	Further Information
Application	An application consists of an executable program that uses the SLEE application API to process events for calls.	*Applications* (on page 4)
Application Instance	An application instance is a running instance of a SLEE application program.	*Application instance* (on page 4)
Service	A service is the functionality provided by an application.	*Service* (on page 4)
Service Handle	A service handle is a name given to the service.	*Service handles* (on page 4)
Service Key	A service key maps to a service or interface.	*Service keys* (on page 5)
Interface	An interface is an executable program that provides a service to the applications.	*Interfaces* (on page 5)
Interface Handle	An interface handle identifies the associated interface.	*Interface handle* (on page 5)
SLEE tools	The SLEE has a set of common tools which can be used to manage the SLEE and the SLEE applications and interfaces.	*SLEE tools* (on page 5)

For a full description of each component, refer to the topics below.

SLEE shared memory

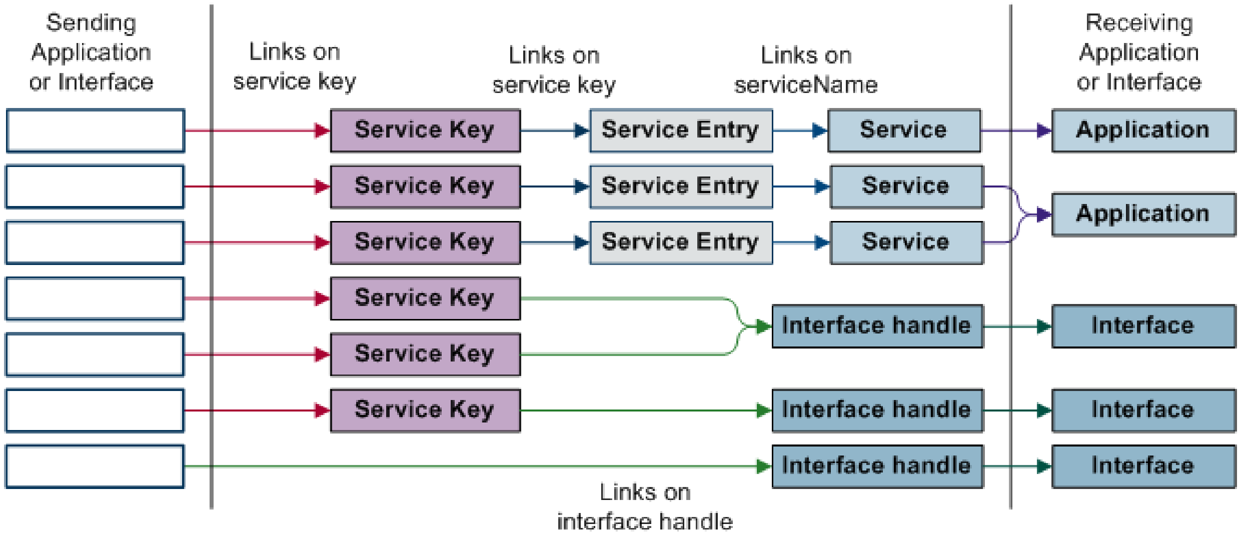
The **/tmp/slee** is the default file used by all SLEE processes at start up to get the shared memory key of the SLEE shared memory. This file must exist and must be the same for all processes wishing to access the same shared memory.

If this file is removed, a SLEE process which is starting up will fail to find shared memory and will exit. If this file is removed, you must restart the SLEE.

For information about overriding the location and filename by setting the environment variable SLEE_FILE for all SLEE processes, see *Optional environmental variables* (on page 10).

Entity relationship

This diagram shows how entries in the **SLEE.cfg** file are related.



Applications

An example of an application is a piece of service logic that provides local number portability processing. An application may support multiple services (for example, it may provide voice VPN in addition to number portability). The SLEE can support multiple different applications simultaneously.

An application is a program that provides a specific set of services to the interfaces. The SLEE allows multiple copies of an application to be started, to enable performance advantages in SMS environments and in cases where an application can block (very briefly) during execution of service logic. This may occur, for example, during an Oracle database read.

Note: Care must be taken to avoid applications from blocking.

Under ideal conditions, an application would never communicate directly with external entities. However, in some cases, this cannot be avoided.

Applications are provided with a set of classes and objects, which provide an interface to the SLEE. All the API functionality for an application is based around call instances. A call instance must be created by an interface (or application acting like an interface). The application's call context memory may also be allocated via the API that is associated with the call instance.

Application instance

Each executing copy of the application is known as an application instance. The SLEE can support multiple running instances of each application program. This is useful in multi-processor environments, where it is possible to be processing events for more than one call in parallel.

Service

Each application may provide multiple services. In the above example, we have an application that provides service and local number portability (SNP & LNP). That application would therefore have two services defined - one for each SNP and LNP.

Service handles

The service handle is defined in the service entity. When a new call is presented to the application, the service handle indicates the particular service for which the call is intended.

Service keys

The service key is the SLEE's mechanism for providing service discrimination. Each service key maps to a service or interface.

Note: A service key is a generic name for the identifiers for a service or interface. Service keys may be derived in some cases from the INAP InitialDP service key.

For more information about configuring service keys, see *SERVICEKEY* (on page 22).

Interfaces

Each service may include an interface to a protocol, which is used to talk to an external entity (SSP, HLR, billing engine, alarm system).

An interface may also generate new calls for the services, applications and application instances.

Examples: Interfaces include:

- TCAP/SS7
- Billing Access
- Databases
- Timers.

Interface handle

Interface handles are used by other elements within the SLEE (other interfaces or application instances) to identify the associated interface.

SLEE tools

The SLEE package also includes a set of tools for managing the SLEE. This table describes these tools.

Tool	Description	Further Information
slee.sh	This script starts the SLEE.	Starting the SLEE.
stop.sh	This script shuts down the SLEE and clears the SLEE shared memory.	Stopping the SLEE.
clean	The clean utility is used to remove SLEE shared memory and semaphores after a unclean SLEE shutdown has occurred.	Removing Shared Memory.
check	This tool provides reports on SLEE attributes.	*check* (on page 29).

SLEE Interfaces

Introduction

An interface is a process that converts messages or events from outside the system from an external format into a common SLEE format. These may then be passed between elements of the SLEE. It also converts SLEE format messages to the external format required. The common format is an object representation of the event called a SLEE Event.

Interfaces are provided with a set of classes and objects, which provide an interface to the SLEE. Interfaces are the main sources of calls requiring processing. This is done by call instances and dialogs with those call instances. A dialog allows messages to be routed between call instances and interfaces. A dialog also allows identification of the call instance or dialog that a SLEE Event can support.

In addition to converting to and from SLEE events and external format events, the interfaces must also respond to SLEE management events. SLEE management events are a specific type of SLEE event.

Types of interface

There are different types of interface. Any interface can be one or more of these types:

Type	Description
Source	Can generate new call instances and dialogs due to external events.
Server	Can only respond to entities that create dialogs with it and make requests (that is, it can only respond to existing call instances).
Sink	A service that never sends a SLEE event. A special method of sending events to sinks is provided that does not use a dialog.

Interface communication through dialogs

While it is normal for interfaces to only communicate through dialogs belonging to call instances, it is also possible for an interface to talk to another interface, through a dialog. It is also possible for an application to talk to another application, through a dialog and call objects.

Example interfaces

This table describes some of the interfaces that may be supplied with the SLEE package.

Interface	Description	Type
timerIF	The Timer interface interacts with the system's real-time clock to provide time-out events to the service logic on demand.	Server
alarmIF	The Alarm interface interacts with the system error logging functionality to report alarms passed from the service logic application.	Sink
statsIF	The Statistics interface maintains and reports statistics to the management system. Statistics are effectively peg counts. statsIF requires a statistics.bin file to run. For more information about creating a statistics.bin file, see *sfVerify* (on page 34).	Sink
replicationIF	The Database Replication interface is an update requester. It processes updates to the system which are generated during call processing. For example, when a client uses call plan functionality to change their profile data. The replicationIF sends an update request to the smsMaster on an SMS. The SMF database is updated with the new information, and is then replicated to all other nodes. For more information about update requests and replication, see *SMS Technical Guide*.	Server/Sink

Interface	Description	Type
cdrIF	The CDR interface writes cdr files containing data received from SLEE CDR events.	Server

Note: Different installations use different SLEE interfaces. If the interface is not listed in the **SLEE.cfg** file, it is not being used by your installation.

Application Programming Interface

Introduction

The SLEE provides an Application Programming Interface (API), through which applications may interact with the SLEE and elements within the SLEE, such as applications and interfaces.

All interactions between applications and other SLEE elements are performed via messages based on objects, which are sub-classes of the SleeEvent class.

The SLEE itself only has one type of SleeEvent, the SleeManagementEvent class. Each of the interfaces provided with the SLEE have their own sub-classes of SleeEvent. SleeEvents may be sent as part of a dialog with another SLEE entity, in which case a SleeDialog object is used to associate SleeEvents of the same dialog. Alternatively, messages may be sent as one-off events, which do not require a response or associated message. In this case, there is no SleeDialog object.

SLEE Dialogs

A SleeDialog provides an association between related messages flowing between SLEE entities (applications and interfaces).

Dialogs also store the 'addresses' of the two entities involved in the dialog. Each dialog has an application side and an interface side.

Note: When an application instance opens a dialog with another application, the first application instance is considered to be the ‘interface’ side of the dialog. Also, if a SLEE interface opens a dialog with another SLEE interface, the first SLEE interface is considered to be the ‘application’ side of the dialog.

Application instances can only open dialogs with interfaces and interfaces can only open dialogs with applications. To enable application to application and interface to interface dialogs, the first application or interface must pretend to be interface or application respectively.

SLEE Events

SLEE events are chunks of shared memory which are used to communicate data within a dialog. Applications and interfaces monitor dialogs for new events.

Management Events

A SLEE management event is an implementation of a SLEE event, which is used to pass management events from the SLEE to applications and interfaces.

The SLEE reserves a number of events user to send management events. These events are added to the first list that has a size greater than, or equal to, 1024 bytes. This means the event count in the check program will show more events than configured in the configuration file.

Supported management events

The following SLEE management events are supported.

Event	Description
WATCHDOG	The watchdog monitors the health of all SLEE elements by sending a series of checks to each configured element. If an element fails to respond, the watchdog will take action to restart the process. Initially, it will send the element a SIGABRT. If the process does not die after a period, a SIGKILL will be sent.
SERVICE_ENABLED	Indicates to an application instance that a service has been enabled. The service handle is passed in the event. Can be used to trigger opening of files, databases, etc.
SERVICE_DISABLED	Indicates to an application instance that a service has been disabled. The service handle is passed in the event. Can be used to trigger closing of files, databases, etc.
APPLICATION_END	Indicates that the application is currently being quiesed and that no new calls will be received. An APPLICATION_KILL will be received when all existing calls complete.
APPLICATION_KILL	Indicates that the application instance should be shutdown and exited. Failure to perform this task within a time period will result in a SIGHTERM being sent to the application instance. If the application instance is still active after a further period, a SIGKILL will be sent.
INTERFACE_END	Indicates that the interface is currently being quiesed and that no new calls will be received or should be generated. An INTERFACE_KILL will be received when all existing dialogs are complete.
INTERFACE_KILL	Indicates that the interface process should be shutdown and exited. Failure to perform this task within a time period will result in a SIGHTERM being sent to the interface process. After a further period, if the interface process is still active, a SIGKILL will be sent.
DIALOG_CLOSED	Indicates to an interface or application that a dialog has been closed by the other end, but no message data has been sent.
REREAD_CONFIG	This message is sent to an application or interface to request it to re-read its configuration. This is a user-management event which the SLEE will never transmit on its own. Not all interfaces support this request.
CALL_INSTANCE_KIL L	The specified call instance has been terminated.
REPORT_REQUEST	This message is sent to an application or interface to generate a short report to stdout or a pre-defined file. Not all interfaces support this request.
CALL_INSTANCE_ TIMED_OUT	The specified call instance has timed out.
DIALOG_TIMED_OUT	The specified dialog has timed out.
STATUS_REQUEST	A status request is sent to an application or interface to generate a short status summary. The receiver should send back a STATUS_RESPONSE message with the current status of the process. Not all interfaces support this request.
STATUS_RESPONSE	Contains the response for the STATUS_REQUEST message.

Chapter 2

Configuration

Overview

Introduction

This chapter explains how to configure the Oracle Communications Network Charging and Control (NCC) application.

In this chapter

This chapter contains the following topics.

Configuration Overview

Introduction

This topic provides a high level overview of how the SLEE is configured.

There are configuration options which are added to the configuration files that are not explained in this chapter. These configuration options are required by the application and should not be changed.

Configuration process overview

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

Step	Action
1	The **SLEE.cfg** file must be configured. This will include configuring: • SLEE maximum values • Watchdog • SLEE interfaces which will be used for this installation (for example, timerIF, cdrIF, and alarmsIF). • SLEE Event Queue Limit
2	Any SLEE interface or application which has an additional configuration file must be configured. For example, the cdrIF is configured using the **cdrIF.cfg** file. **Note:** Most installations will require other applications and interfaces to be configured in the **SLEE.cfg** also. This should be done after the other applications have been installed. For more information about how to configure additional interfaces and applications, see the documentation for the application.

Configuration components

The SLEE is configured by the following components:

Component	Locations	Description	Further Information
SLEE.cfg	all machines	The only file used to configure the SLEE is **SLEE.cfg**. The configuration file is used to configure the applications, services and interfaces which the SLEE manager will initialize. From this information the SLEE manager also knows how much shared memory to allocate. **SLEE.cfg** is broken into three sections: • Maximum object instances • Application entries • Interface entries	*Configuring the SLEE* (on page 10).
Environmental variables	all machines	SLEE supports some environmental variables.	*Optional environmental variables* (on page 10)
SLEE_callID.cfg	all machines	This file configures the last written SLEE call ID value when the SLEE is shut down. On start up, SLEE reads this file's value and adds a 1000 to it. This guarantees that the next generated EDR call ID is not a duplicate.	*Configuring the SLEE Call ID*
cdrIF.cfg	all machines	This file configures the cdrIF.	
tcRelayMappings.def	all machines	This file configures the tcRelayApp.	*Configuration* (on page 40)

Configuring the SLEE

Introduction

The SLEE must be configured at start-up. The default configuration file is **/IN/service_packages/SLEE/etc/SLEE.cfg**.

Optional environmental variables

SLEE supports the following environmental variables:

`SLEE_FILE`

Syntax:	
Description:	The location of the file which stores the shared memory keys for the SLEE's shared memory.
Type:	
Optionality:	Optional.
Allowed:	
Default:	/IN/service_packages/SLEE/tmp/slee_file
Notes:	All SLEE processes must use the same SLEE_FILE.
Example:	

Modifying SLEE.cfg

Each section of the **SLEE.cfg** configuration file is detailed below, with examples of appropriate settings.

The SLEE must be restarted for any configuration changes to take effect. For more information on restarting the SLEE, see *Tools and Utilities* (on page 27).

Setting a SLEE Event Queue Limit

You can configure the SLEE to prevent new dialogs and calls from being created when a large queue of events are waiting to be processed. Existing calls and events are still processed. This configuration enables an overloaded system to reject calls instead of denying services.

You can set a limit to the number of events that can be queued on any SLEE interface or application. If any SLEE interface or application exceeds that number in their queue, all SLEE interfaces and applications are not allowed to create dialogs to create new SLEE calls. The interfaces and applications will continue to process existing calls.

For example, if you set a limit of 60, when the Session Control Agent (SCA) queue grows beyond 60, no more events are queued to the SCA, or to the Open Services Development (OSD) and Diameter Control Agent (DCA) queues.

To configure the SLEE event limit, edit the `SLEEWIDEEVENTLIMIT` in the **SLEE.cfg** file. For example:

```
SLEEWIDEEVENTLIMIT=60
```

The value can be from 1 to 4294967295. The default is 4294967295.

The watchdog gives the following alarms, no more than one per second:

- When all queues are below 80% of the allowed maximum, for example: `Oct 19 21:28:08.005983 watchdog(27270) NOTICE: Highest number of events queued on any one process is now below 80% of allowed maximum. Allowed events: 60, current events: 42, process: slee_acs.sh`
- When a queue is between 80% and 90% of the allowed maximum, for example:`Oct 19 21:27:54.007430 watchdog(27270) WARNING: Highest number of events queued on any one process is now between 80% and 90% of allowed maximum. Allowed events: 60, current events: 49, process: slee_acs.sh`
- When a queue exceeds 90% of the allowed maximum, for example:`Oct 19 21:29:32.006240 watchdog(27270) ERROR: Highest number of events queued on any one process is now above 90% of allowed maximum. Allowed events: 60, current events: 61, process: slee_acs.sh`

You can get information about the number of queued SLEE events by using option **a** in the check utility; for example:

```
Select:  a
Name              A/I Status  Events  Limit   Total
sleeTrafficCDMA   I   START   0       none    0
sca               I   ACTIVE  0       none    2
smsc              I   ACTIVE  0       none    2
xmsStoreIf        I   ACTIVE  0       none    2
xmsAgentIf        I   ACTIVE  0       none    2
xmsIf             I   ACTIVE  0       none    2
ScriptTcIF        I   ACTIVE  0       none    2
osd               I   ACTIVE  0       none    2
dcaIf             I   ERROR   0       none    0
slpitCAP4_2       I   START   0       none    0
slpitCAP4         I   START   0       none    0
slpitCS1          I   START   0       none    0
dcdBeClient       I   ACTIVE  0       none    2
rimsIf            I   ACTIVE  0       none    2
```

```
acsStatsLocalSLEEI   START   0       none    0
Timer            I   ACTIVE  0       none    2
CdmaGw           I   ACTIVE  0       none    2
sua_if1          I   ACTIVE  0       none    2
m3ua_if2         I   ACTIVE  0       none    2
m3ua_if1         I   ACTIVE  0       none    2
textIF           I   ACTIVE  0       none    2
cdmaTraffic      I   ACTIVE  0       none    2
vssp             I   ACTIVE  0       none    2
tcRelayApp.sh    A   ACTIVE  0       none    0
ussdgw.sh        A   ACTIVE  0       none    0
slee_acs.sh      A   ACTIVE  0       none    224
slee_acs.sh      A   ACTIVE  0       none    226
Maximum events queued on one process:
61 events on slee_acs.sh at Sun Oct 19 21:29:31 2014
```

When the SLEE event queue limit is reached, the individual SLEE applications prevent system overloads by being marked as overloaded. The applications then use their own overload mechanisms to reject new messages and report errors. For example, when the SLEE event queue limit is reached, the Open Services Development (OSD) application uses the osdInterface to reject new messages and return errors.

For example,

```
Oct 19 22:51:27.418425 osdInterface(18297) WARNING: {1100016} osdInterface is marked
as overloaded
```

Setting Up SLEE to Generate a Separate Log File for Each Application Instance

You can configure SLEE to generate a separate log file for each SLEE application instance. Separate log files help you to identify the SLEE instance used for the call while analyzing and debugging the call information.

To set up SLEE to generate a separate log file for each application instance:

Step	Action
1.	Update the following in slee.sh: `export APP_INSTANCE_PID_EXPORT="yes"`

2.	Add the following in slee_acs.sh:
	`if [ -n "$APP_INSTANCE" ]; then`
	`LOGFILE_INSTANCE=${SEPERATOR}$APP_INSTANCE`
	`fi`

3.	(Optional) To add PID to the file name, add the following in slee_acs.sh:
	`LOGFILE_PID=""`
	`SEPERATOR="-"`
	`if [ -n "$1" ]; then`
	`LOGFILE_PID="${SEPERATOR}$1"`
	`fi`

4.	Edit the exec line in slee_acs.sh to include the LOGFILE_INSTANCE and LOGFILE_PID:
	`exec /IN/service_packages/ACS/bin/slee_acs >>`
	`/IN/service_packages/ACS/tmp/slee_acs${LOGFILE_INSTANCE}${`
	`LOGFILE_PID).log`
	`2>&1`

Examples of output log files:

```
bash-3.00$ ls -lct |head
```

Log file name with instance number:

```
-rw-r--r-- 1 acs_oper esg 62281 Jul 30 12:57 slee_acs01.log
-rw-r--r-- 1 acs_oper esg 62281 Jul 30 12:57 slee_acs04.log
-rw-r--r-- 1 acs_oper esg 62281 Jul 30 12:57 slee_acs02.log
```

```
-rw-r--r-- 1 acs_oper esg 62281 Jul 30 12:57 slee_acs03.log
```

Log file name with instance number and PID:

```
-rw-r--r-- 1 acs_oper esg 62406 Jul 30 13:07 slee_acs-01-26030.log
-rw-r--r-- 1 acs_oper esg 62281 Jul 30 13:05 slee_acs-03-26032.log
-rw-r--r-- 1 acs_oper esg 62281 Jul 30 13:05 slee_acs-02-26031.log
-rw-r--r-- 1 acs_oper esg 62281 Jul 30 13:05 slee_acs-04-26033.log
```

Maximum values

The following maximum lengths apply to names used in the **SLEE.cfg** file:

- 20 characters for interface names
- 20 characters for service names
- 40 characters for application names

The first section of **SLEE.cfg** contains the maximum number of each type of object which can be held in shared memory.   If this value is exceeded, an exception will be thrown and an entry made in the syslog. In many cases this will cause the SLEE and all processes to restart.

Format:

```
MAXAPPLICATIONS=max
MAXSERVICES=max
MAXSERVICEHANDLES=max
MAXSERVICEKEYS=max
MAXDIALOGS=max
MAXEVENTS=max [size]
MAXCALLS=max
MAXINTERFACES=max
MAXEVENTTYPES=max
```

The available parameters are:

MAXAPPLICATIONS

Syntax:	`MAXAPPLICATIONS=`*`max`*
Description:	The maximum number of application objects which can be held in shared memory.
Type:	Integer
Optionality:	Mandatory
Allowed:	
Default:	none
Notes:	This also sets the maximum number of *APPLICATION* (on page 23) entries which can be active in the **SLEE.cfg** file.
Example:	`MAXAPPLICATIONS=5`

MAXSERVICES

Syntax:	`MAXSERVICES=`*`max`*
Description:	The maximum number of service objects which can be held in shared memory.
Type:	Integer
Optionality:	Mandatory
Allowed:	
Default:	none

Notes:	This also sets the maximum number of *SERVICE entries* (on page 20) which can be active in the **SLEE.cfg** file.
Example:	`MAXSERVICES=5`

MAXSERVICEHANDLES

Syntax:	`MAXSERVICEHANDLES=`*`max`*
Description:	The maximum number of service handles.
Type:	Integer
Optionality:	Optional (default used if not set).
Allowed:	
Default:	10
Notes:	MAXSERVICEHANDLES has to be greater than or equal to the number of distinct service handles specified in *SERVICE entries* (on page 20) in **SLEE.cfg**
Example:	`MAXSERVICEHANDLES=20`

MAXSERVICEKEYS

Syntax:	`MAXSERVICEKEYS=`*`max`*
Description:	The maximum number of service key objects which can be held in shared memory.
Type:	Integer
Optionality:	Optional (default used if not set).
Allowed:	
Default:	If this parameter is not specified, the count of serviceKey entries in the configuration files are used.
Notes:	This also sets the maximum number of *SERVICEKEY* (on page 22) entries which can be active in the **SLEE.cfg** file.
Example:	`MAXSERVICEKEYS=5`

MAXDIALOGS

Syntax:	`MAXDIALOGS=`*`max`*
Description:	The maximum number of dialog objects which can be held in shared memory.
Type:	Integer
Optionality:	Mandatory
Allowed:	
Default:	none
Notes:	
Example:	`MAXDIALOGS=5`

MAXEVENTS

Syntax:	`MAXEVENTS=`*`max size`*
Description:	The maximum number of event objects which can be held in shared memory.
Type:	Integer
Optionality:	Mandatory
Allowed:	
Default:	none

Notes:	MAXEVENTS can be specified more than once for multiple lists of differing sizes and the effects are cumulative. This list will then have its count modified (when the SLEE starts up) to handle the management event reservation. This affects the check tool event count. For more information, see *check* (on page 29). If the MAXEVENTS values are exceeded when the system is running, no more events or calls will be accepted and alarm messages will be sent.
Example:	To get 500 events of size 2k, and 200 of size 4k, set: `MAXEVENTS=500 2048` `MAXEVENTS=200 4096`

size

Syntax:	*`size`*
Description:	*MAXEVENTS* (on page 14). supports the *`size`* parameter which allows you to set the maximum data segment size of an event.
Type:	Integer
Optionality:	Optional (default used if not set).
Allowed:	
Default:	1024
Notes:	There must be at least one list with a size equal to or greater than 1024 bytes.
Example:	For an example of this parameter in context, see *MAXEVENTS* (on page 14).

MAXCALLS

Syntax:	`MAXCALLS=`*`max`*
Description:	The maximum number of call objects which can be held in shared memory.
Type:	Integer
Optionality:	Mandatory
Allowed:	
Default:	none
Notes:	If the MAXCALLS values are exceeded when the system is running, no more events or calls will be accepted and alarm messages will be sent.
Example:	`MAXCALLS=500`

MAXINTERFACES

Syntax:	`MAXINTERFACES=`*`max`*
Description:	The maximum number of interface objects which can be held in shared memory.
Type:	Integer
Optionality:	Mandatory
Allowed:	
Default:	
Notes:	This also sets the maximum number of INTERFACE entries which can be active in the **SLEE.cfg** file.
Example:	`MAXINTERFACES=5`

MAXEVENTTYPES

Syntax:	`MAXEVENTTYPES=`*`max`*
Description:	The maximum number of Event Type objects which can be held in shared memory.
Type:	Integer
Optionality:	Mandatory
Allowed:	
Default:	
Notes:	
Example:	`MAXEVENTTYPES=5`

APPENDINTERVAL

Syntax:	`APPENDINTERVAL=<secs>`
Description:	The delay/interval between ending SLEE interfaces and SLEE applications during the SLEE stop operation.
Type:	Integer
Optionality:	Optional (default 0 if not set).
Allowed:	
Default:	0
Notes:	
Example:	`APPENDINTERVAL=2`

Other parameters

These parameters are set after the Maximum values configuration in **SLEE.cfg**.

NOWIPESOCKETS

Syntax:	`NOWIPESOCKETS=`*`int`*	
Description:	How to handle pre-existing sockets on startup.	
Type:	Integer	
Optionality:	Optional (default used if not set).	
Allowed:	0	Socket files corresponding to **SLEE.cfg**-specified interfaces are removed.
	anything else	No socket files will be removed.
Default:	0	
Notes:	This setting is designed for use in testing environments where more than one SLEE is running concurrently. It should not be used in production.	
Example:		

SLEEWIDEEVENTLIMIT

Syntax:	`SLEEWIDEEVENTLIMIT = `*`Events`*
Description:	Sets the limit of events for SLEE calls, after which the SLEE does not create dialogs for new SLEE calls.
Type:	Integer
Optionality:	Optional (default used if not set)
Allowed:	1 to 4294967295
Default:	4294967295

Notes:	If more than SLEEWIDEEVENTLIMIT SLEE events are queued on any one SLEE interface or any one SLEE application instance, then all SLEE applications and SLEE Interfaces will not be able to create dialogs that create new SLEE calls.
Example:	`SLEEWIDEEVENTLIMIT = 60`

INTERFACE

You use the INTERFACE entries to configure the interfaces you want the SLEE to run.   You can run multiple instances of the same interface. You specify the number of instances of a particular interface to run in an INTERFACE entry parameter. Messages can be sent directly to an interface handle, or to a service key.

Installing and removing packages may add or remove interface entries. Do not manually remove entries from the **SLEE.cfg** file that have been added by the package installation or removal process.

Usage:

```
INTERFACE=interface_handle interface_name interface_path instance_count
interface_type [int_event_count] [dialog_count] [Suspect Cycles = n] [Throttle
Cycles = n]
```

INTERFACE parameters

`interface_handle`

Syntax:	To see this parameter in context, see *Usage* (on page 17).
Description:	The unique name that identifies this SLEE interface. You can run multiple instances of a particular interface by specifying the number of instances to run in the *instance_count* (on page 18) parameter.
Type:	String
Optionality:	Required.
Allowed:	The last character in the name should not be numeric.
Default:	None
Notes:	If you configure the SLEE to run multiple instances of an interface, the SLEE appends a unique number to each instance of the interface handle, at startup. The system also creates a separate log file for each interface instance and appends the number of the interface instance to the log file name.
Example:	For an example of this parameter used in context, see *Example INTERFACE Configuration* (on page 20).

`interface_name`

Syntax:	To see this parameter in context, see *Usage* (on page 17).
Description:	The name of the executable file that enables this interface.
Type:	String
Optionality:	Required
Allowed:	A valid NCC binary.
Default:	None
Notes:	
Example:	For an example of this parameter used in context, see *Example INTERFACE Configuration* (on page 20).

interface_path

Syntax:	To see this parameter in context, see *Usage* (on page 17).
Description:	The full path to the interface binary file.
Type:	String
Optionality:	Required
Allowed:	A valid path.
Default:	None
Notes:	
Example:	For an example of this parameter used in context, see *Example INTERFACE Configuration* (on page 20).

instance_count

Syntax:	To see this parameter in context, see *Usage* (on page 17).
Description:	The number of instances to run of the defined SLEE interface.
Type:	Integer
Optionality:	Optional (default used if not set).
Allowed:	A numeric value that is greater than or equal to 1(one).
Default:	1
Notes:	If you configure to run more than one instance of a particular interface, then the SLEE creates a unique ID for each instance of the interface by appending a number to the interface name. The number will be in the range 0 to *n*-1, where *n* is the configured *instance_count* value. A number will not be appended to the interface name if there is only one instance of the interface.
Example:	For an example of the parameter used in context, see *Example INTERFACE Configuration* (on page 20).

interface_type

Syntax:	To see this parameter in context, see *Usage* (on page 17).
Description:	The type of interface.
Type:	String
Optionality:	Required
Allowed:	EVENT
	UGD
Default:	None
Notes:	
Example:	For an example of this parameter used in context, see *Example INTERFACE Configuration* (on page 20).

int_event_count

Syntax:	To see this parameter in context, see *Usage* (on page 17).
Description:	The maximum number of events allowed to wait for processing before call limiting for the interface starts.
Type:	Integer
Optionality:	Optional, but required if *dialog_count* is set.
Allowed:	
Default:	Unlimited

Notes:	If the number of events exceeds the limit specified in the configuration file, then any further attempts to create another dialog to the interface will fail. This failure will then be handled by the process attempting to create the dialog.
Example:	For an example of this parameter used in context, see *Example INTERFACE Configuration* (on page 20).

`dialog_count`

Syntax:	To see this parameter in context, see *Usage* (on page 17).
Description:	The maximum number of dialogs that can be open on the interface.
Type:	Integer
Optionality:	Optional (default used if not set).
Allowed:	
Default:	1
Notes:	The SLEE tracks the number of dialogs open on the interface. If the interface has exceeded the limit specified in the configuration file, any further attempts to create a dialog on the interface will fail.
Example:	For an example of this parameter used in context, see *Example INTERFACE Configuration* (on page 20).

`Suspect Cycles`

Syntax:	To see this parameter in context, see *INTERFACE* (on page 17).
Description:	Specifies the number of watchdog cycles to wait before restarting the interface, in the event that it becomes non-responsive. Use this parameter to allow the interface time to finish dumping core for diagnostic purposes.
Type:	Integer
Optionality:	Optional
Allowed:	0 to maximum integer value (2,147,483,647)
Default:	0
Notes:	If greater than 0, overrides WATCHDOGSUSPECTCYCLES.
Example:	Suspect Cycles=2

`Throttle Cycles`

Syntax:	To see this parameter in context, see *INTERFACE* (on page 17).
Description:	Specifies the number of watchdog cycles to use for the throttle period (Throttle Cycles times WATCHDOGCYCLETIME), which begins when an interface starts. If an interface dies or becomes unresponsive *during* the throttle period, the watchdog process restarts it after an additional watchdog cycle. This prevents the watchdog process from continuously restarting processes that die immediately due to incorrect configuration, for example. If the interface dies or becomes unresponsive *after* the throttle period, the watchdog process treats it like any other unresponsive process.
Type:	Integer
Optionality:	Optional
Allowed:	0 to maximum integer value (2,147,483,647)
Default:	0
Notes:	If greater than 0, overrides WATCHDOGTHROTTLECYCLES.
Example:	Throttle Cycles=2

Example INTERFACE Configuration

These lines in **SLEE.cfg** configure three timer interface instances, one replication interface instance, and two notificationIF interface instances:

```
INTERFACE=Timer timerIF /IN/service_packages/SLEE/bin 3 EVENT
INTERFACE=Replication replicationIF.sh /IN/service_packages/SLEE/bin 1 EVENT
INTERFACE=notificationIF notificationIF /IN/service_packages/SLEE/bin 2 UDG
```

At startup, the SLEE creates a unique ID for each instance of an interface by appending a number in the range 0 to *n*-1 to the interface name, where *n* is the number of interface instances configured in the *instance_count* (on page 18) parameter. If there is only one instance of an interface, then the SLEE does not append a number to the interface name. This means that the three `Timer` interface instances in the example would have the following IDs: `Timer0`, `Timer1`, and `Timer2`. Whereas the ID for the single instance of the `Replication` interface would be: `Replication`.

watchdog

This section defines the location and cycle time for the watchdog.   You should not need to alter these settings.

```
WATCHDOG=/IN/service_packages/SLEE/bin/ watchdog
WATCHDOGCYCLETIME=30
```

You can also specify values for WATCHDOGSUSPECTCYCLES and WATCHDOGTHROTTLECYCLES parameters. For more information, see *watchdog* (on page 41).

SERVICE entries

The SERVICE entries define each service object to be created in shared memory.   The service name, priority and the name of the application that provides this service are defined here.   Each service must be associated with an application.

Note: You cannot have more SERVICES than the number allowed by *MAXSERVICEHANDLES* (on page 14).

Usage:

```
SERVICE=serviceName priority appName serviceHandle [callCount]
```

The available parameters are:

serviceName

Syntax:	To see this parameter in context, see *SERVICE entries* (on page 20).
Description:	The name of the service provided by an application.
Type:	String
Optionality:	Mandatory
Allowed:	
Default:	
Notes:	This matches the *dest* (on page 22) parameter in the *SERVICEKEY* (on page 22) entry for the service keys which will use this service. It must match at least one service key entry.
Example:	For an example of this parameter used in context, see *Examples* (on page 22).

priority

Syntax:	To see this parameter in context, see *SERVICE entries* (on page 20).
Description:	The priority the scheduler gives this service.
Type:	Integer
Optionality:	Mandatory

Allowed:	
Default:	
Notes:	
Example:	For an example of this parameter used in context, see *Examples* (on page 22).

appName

Syntax:	To see this parameter in context, see *SERVICE entries* (on page 20).
Description:	The name of the application which enables this service.
Type:	String
Optionality:	Mandatory
Allowed:	
Default:	
Notes:	This matches to the *appName* (on page 23) parameter in the *APPLICATION* (on page 23) for the application which will handle this service. It must match at least one application entry.
Example:	For an example of this parameter used in context, see *Examples* (on page 22).

serviceHandle

Syntax:	To see this parameter in context, see *SERVICE entries* (on page 20).
Description:	The service handle which is sent to the application to enable it to provide more than one service.
Type:	String
Optionality:	Mandatory
Allowed:	
Default:	
Notes:	It must match the service handle defined in the application entry this service entry links to. **Example:** serviceHandles for slee_acs must match serviceNames in ServiceEntries in **acs.conf**. For more information about ACS ServiceEntries, see *ACS Technical Guide*. There will typically be multiple lines of this type for each appName as one application will usually handle more than one service.
Example:	For an example of this parameter used in context, see *Examples* (on page 22).

callCount

Syntax:	To see this parameter in context, see *SERVICE entries* (on page 20).
Description:	The maximum number of concurrent calls which can be processed by this service.
Type:	Integer
Optionality:	Optional
Allowed:	
Default:	unlimited
Notes:	If the number of calls active on this service exceeds the specified limit, all attempts to create a call for this service will fail.
Example:	For an example of this parameter used in context, see *Examples* (on page 22).

Examples

This text shows some examples of SERVICE entries in a **SLEE.cfg** file.

```
SERVICE=PREPAID 1 slee_acs CCS
SERVICE=ACS_Outgoing 1 slee_acs ACS_Outgoing
```

SERVICEKEY

The service key entries define each service key. They also include information on which service or interface will handle this service key. Each service key must be associated with either a service or an interface instance.

Service keys have the following configuration options:

SERVICEKEY=*keyType serviceKey dest*

The available parameters are:

keyType

Syntax:	To see this parameter in context, see *SERVICEKEY* (on page 22).
Description:	The type of service key.
Type:	Integer
Optionality:	
Allowed:	
Default:	
Notes:	
Example:	For an example of this parameter used in context, see *Examples* (on page 23).

serviceKey

Syntax:	To see this parameter in context, see *SERVICEKEY* (on page 22).
Description:	The service key from interface.
Type:	
Optionality:	
Allowed:	Format depends on interface.
Default:	
Notes:	
Example:	For an example of this parameter used in context, see *Examples* (on page 23).

dest

Syntax:	To see this parameter in context, see *SERVICEKEY* (on page 22).
Description:	Service name or interface name.
Type:	
Optionality:	
Allowed:	
Default:	
Notes:	Must match a *serviceName* (on page 20) or ifHandle.
Example:	For an example of this parameter used in context, see *Examples* (on page 23).

Examples

This text shows examples of SERVICEKEY entries.

```
SERVICEKEY=INTEGER 101 PREPAID
SERVICEKEY=INTEGER 1 0800
```

The serviceKey depicts the service key that this application will handle. There will typically be multiple lines of this type for each appName as one application will usually handle more than one service key.

APPLICATION

The application entry enables the SLEE to run the binary files.

Usage:

```
APPLICATION=appName execName execDir startInstances maxInstances [appEventCount]
[Suspect Cycles] [Throttle Cycles]
```

The available parameters are:

`appName`

Syntax:	To see this parameter in context, see *APPLICATION* (on page 23).
Description:	The name of the application.
Type:	String
Optionality:	Mandatory
Allowed:	
Default:	none
Notes:	This is used to refer to the application in other parts of the configuration file, including *SERVICE entries* (on page 20), where it must be matched by *appName* (on page 21).
Example:	For an example of this parameter used in context, see *Example* (on page 25).

`execName`

Syntax:	To see this parameter in context, see *APPLICATION* (on page 23).
Description:	The name of the binary file to be run.
Type:	String
Optionality:	Mandatory
Allowed:	
Default:	none
Notes:	
Example:	For an example of this parameter used in context, see *Example* (on page 25).

`execDir`

Syntax:	To see this parameter in context, see *APPLICATION* (on page 23).
Description:	Full path to the directory where the executable binary is stored.
Type:	String
Optionality:	Optional (default used if not set).
Allowed:	Any directory path.
Default:	"/IN/service_packages/SLEE/bin"
Notes:	
Example:	For an example of this parameter used in context, see *Example* (on page 25).

`startInstances`

Syntax:	To see this parameter in context, see *APPLICATION* (on page 23).
Description:	The number of instances of the application the SLEE should initially start.
Type:	Integer
Optionality:	Optional (default used if not set).
Allowed:	
Default:	1
Notes:	
Example:	For an example of this parameter used in context, see *Example* (on page 25).

`maxInstances`

Syntax:	To see this parameter in context, see *APPLICATION* (on page 23).
Description:	The maximum number of instances of the application that the SLEE will support.
Type:	Integer
Optionality:	Optional (default used if not set).
Allowed:	
Default:	1
Notes:	
Example:	For an example of this parameter used in context, see *Example* (on page 25).

`appEventCount`

Syntax:	To see this parameter in context, see *APPLICATION* (on page 23).
Description:	The maximum number of events allowed to be awaiting processing before call limiting for the application starts.
Type:	Integer
Optionality:	Optional
Allowed:	
Default:	
Notes:	• `app event count` applies to the application as a whole, that is, all the application instances combined. If `app event count` is set to 1000 and `start num instance` and `max num instance` are both set to 2, then two application instance processes will run and each one can have up to 500 events queued. • If the number of events exceeds the limit specified in the configuration file, then any further attempts to create another call instance will fail.
Example:	For an example of this parameter used in context, see *Example* (on page 25).

`Suspect Cycles`

Syntax:	To see this parameter in context, see *APPLICATION* (on page 23).
Description:	Specifies the number of watchdog cycles to wait before restarting the application in the event that it becomes unresponsive. Use this parameter to allow the application time to finish dumping core for diagnostic purposes.
Type:	Integer
Optionality:	Optional
Allowed:	0 to maximum integer value (2,147,483,647)
Default:	0

Notes:	If greater than 0, overrides WATCHDOGSUSPECTCYCLES
Example:	Suspect Cycles=2

Throttle Cycles

Syntax:	To see this parameter in context, see *APPLICATION* (on page 23).
Description:	Specifies the number of watchdog cycles to use for the throttle period (Throttle Cycles times WATCHDOGCYCLETIME), which begins when an application starts. If an application dies or becomes unresponsive *during* the throttle period, the watchdog process restarts it after an additional watchdog cycle. This prevents the watchdog process from continuously restarting processes that die immediately due to incorrect configuration, for example. If the application dies or becomes unresponsive *after* the throttle period, the watchdog process treats it like any other unresponsive process.
Type:	Integer
Optionality:	Optional
Allowed:	0 to maximum integer value (2,147,483,647)
Default:	0
Notes:	If greater than 0, overrides WATCHDOGTHROTTLECYCLES
Example:	Throttle Cycles=2

Example

This text shows an example of an APPLICATION entry.

```
APPLICATION=appExample appExample ../appExample 1 1
```

Example SLEE.cfg file

This is an example of the configuration part of a **SLEE.cfg** file.

```
MAXAPPLICATIONS=10
MAXSERVICES=10
MAXSERVICEHANDLES=10
MAXSERVICEKEYS=20
MAXDIALOGS=70000
MAXEVENTS=50000
MAXCALLS=25000
MAXINTERFACES=20
MAXEVENTTYPES=30
MAXCORRELATIONIDS=10000
SLEEWIDEEVENT=60

INTERFACE=Timer timerIF /IN/service_packages/SLEE/bin 1 EVENT
INTERFACE=acsStatsLocalSLEE acsStatsLocalSLEE /IN/service_packages/ACS/bin 1 EVENT
INTERFACE=Replication replicationIF.sh /IN/service_packages/SLEE/bin 1 EVENT
INTERFACE=hssScIf hssScIf.sh /IN/service_packages/SLEE/bin 1 EVENT

WATCHDOG=/IN/service_packages/SLEE/bin/ watchdog
WATCHDOGCYCLETIME=30
WATCHDOGSUSPECTCYCLES=2
WATCHDOGTHROTTLECYCLES=3
APPENDINTERVAL=2

# SLEE Process Manager (statistics collection)
#INTERFACE=sleeProcMan sleeProcMan /IN/service_packages/SLEE/bin 1 UDG

# APPLICATION
```

```
APPLICATION=mngApp mngApp /IN/service_packages/SLEE/bin 1 1

# SERVICE
SERVICE=ACS 1 slee_acs ACS
SERVICE=ACS_Outgoing 1 slee_acs ACS_Outgoing

# SERVICEKEY
SERVICEKEY=INTEGER 111 ACS
SERVICEKEY=INTEGER 110 ACS_Outgoing
```

Configuring the SLEE Call ID

Introduction

The SLEE call ID must be configured at start-up. The default configuration file is **/IN/service_packages/SLEE/etc/SLEE_callID.cfg**.

You can change this by setting the SLEE_FILE environmental variable.

Parameters

The following configuration parameters are provided to control the SLEE call ID:

`GLOBALCIDWRITEFREQ`

Syntax:	`GLOBALCIDWRITEFREQ = `*`num`*
Description:	The global call ID write frequency. Specifies how often to write the SLEE's current call ID to the **SLEE_callID.cfg** file.
Type:	Integer
Optionality:	Optional (default used if not set)
Allowed:	A value of 0 or greater: • 0 – The value of 1000 is added to the value in **SLEE_callID.cfg** on SLEE start up. • Greater than 0 – The value of `GLOBALCIDWRITEFREQ` is added to the **SLEE_callID.cfg** value on SLEE startup.
Default:	0
Notes:	
Example:	`GLOBALCIDWRITEFREQ = 100`

Chapter 3

Tools and Utilities

Overview

Introduction

This chapter explains how to use the utilities provided with the SLEE. To:

- Start the SLEE, see *stop.sh* (on page 28).
- Shut down the SLEE, see *stop.sh* (on page 28).
- Remove Shared Memory and semaphores, see *clean* (on page 28).
- Display what SLEE resources are in use, see *check* (on page 29).
- Create a **statistics.bin** file for statsIF, see *sfVerify* (on page 34).

Warning: All these scripts must be run from **/IN/service_packages/SLEE/bin**. Unpredictable results will occur if run from elsewhere.

In this chapter

This chapter contains the following topics.

slee.sh

Purpose

slee.sh provides a standardized way of starting the SLEE.

Warning: Running this script while the SLEE is already running will result in the SLEE becoming unstable.

Configuration

The slee.sh does not support any configuration options.

Failure

If slee.sh fails, the SLEE will not start properly. To ensure you start the SLEE in a stable environment, complete the following before you run slee.sh again:

- Run stop.sh
- Run clean
- Ensure all SLEE processes are terminated

Output

slee.sh writes error messages to the system messages file.

stop.sh

Purpose

The stop.sh script shuts down the SLEE in a controlled manner. This ensures the SLEE shared memory and semaphores are cleared.

Configuration

stop.sh does not support any configuration options.

Failure

If the stop.sh script has failed, the SLEE may not have been shut down properly. Attempt to run the stop.sh script again, and run clean to ensure all SLEE shared memory has been properly removed and all processes have been removed.

Output

stop.sh writes error messages to the system messages file.

clean

Purpose

The clean tool uses the Unix clean tool to remove an current SLEE shared memory and semaphores. This must be completed if the SLEE has exited without being shut down properly, for example if there was a network outage.

Startup

clean is started by acs_oper from the command line using the following command:

```
/IN/service_packages/SLEE/bin/clean
```

Configuration

clean does not support any configuration options.

Failure

If clean fails, SLEE Shared Memory may still exist. Attempt to rerun the script.

Output

clean writes error messages to the system messages file.

check

Purpose

The check utility provides a method of monitoring the SLEE, verifying start-up, and analyzing the SLEE configuration file, **SLEE.cfg**. It can produce either periodic reports or general reports.

About Reported Events

The total number of events reported by check will not match the total event number specified in the configuration file. This is because SLEE reserves a set of events for exclusive use by the watchdog process. These events are used to clean up the SLEE if a runaway process allocates all the available SLEE resources and needs to be cleaned up. The additional number of events is calculated as:

```
Extra Events = (Max dialogs * 2) + (Max application instances * 2) + (Max interfaces
* 2)
```

These events are added into the list with a size greater than, or equal to, the default of 1024.

Configuration

check supports several different usage modes with different parameters available to each mode. For information about:

- Checking the SLEE in batch mode, see *Checking the SLEE by Using Batch Mode* (on page 29).
- Confirming if SLEE processes are running, see *Confirming if SLEE Processes Are Running* (on page 30).
- Getting parsing information, see *Getting Parsing Information on SLEE Processes* (on page 30).
- Checking correlation IDs, see *Checking Correlation IDs* (on page 31).
- Testing event consumption, see *Testing Event Consumption, Spillage, and Leakage* (on page 32).

For information about check menu options, see *Main Menu Options* (on page 32).

Checking the SLEE by Using Batch Mode

check supports the following command-line parameters in batch mode:

```
check [mode] [-S] [-Q] [-iN_items] [-pPID] [-tevent_type] [[-u] [-m] [-z]]
[output_mode]
```

where:

- *mode* is one of the following options: `-c`, `-C`, `-d`, `-D`, `-e`, `-E`, `-f`, or `-q`. The mode is set to `-f` by default.
- *output_mode* is the following: [`-v`] [`-n`*`N_items`*] [`-b`]|[`-B`] *`delay`* [*`count`*]

The following table describes the available parameters. The parameters appear in the table in order of usage.

Parameter	Description
mode	Display information about SLEE processes where *mode* specifies the type of information to display in the report: Specify one of: `-c`, to include floating calls `-C`, to display all calls

Parameter	Description
	`-d`, to display active dialogs `-D`, to display all dialogs `-e`, to display floating events `-E`, to display all events `-f`, to display free object counts `-q`, to display event queues If not specified, *mode* defaults to **`-f`**.
-S	Exit the check utility if the SLEE connection fails.
-Q	Quiet the logging of exceptions into messages log (turn off logging messages into messages log).
-i*`N_items`*	Display the state of each of the listed correlation IDs used by specialized resource functions (SRFs) such as IVRs. The check report will pause after *`N_items`* rows. If you set *`N_items`* to 0 (zero), the check report lists the state of all the correlation IDs.
-p*PID*	Specify with the `-E` option. Filter for events with the PID number specified in *PID*.
-t*event_type*	Specify with the `-E` option. Filter for events with the event type specified in *event_type*.
-u	Specify with the `-f` option. Display the check report in table format. Use the COLUMN_WIDTH SLEE environment variable to set the column width in the report.
-m	Specify with the `-u` option. Display the maximum of each object type used or over-spilled instead of the current values.
-z	Specify with the `-u` option. Display only free event object counts; for example, do not display calls or dialogs.
`-v`	Verbose. If specified with the `-E` *mode*, include data in event dumps and print the SLEE connect and disconnect times.
-n*`N_items`*	Display the top *`N_items`* items when specified with the `-f`, `-q`, or `-E` *mode*. When specified with: • `-f` or `-q`, *`N_items`* is the number of rows before header insertion • `-E`, limit the number of events displayed to between 1 and *`N_items`* items
`-b`	Print the report header only once. Specify `-b` or `-B`, but not both.
-B	Do not print the report header. Specify `-b` or `-B`, but not both.
`delay`	The number of seconds before the specified report is repeated.
`count`	The number of times the report should be repeated. If not specified, *`count`* defaults to 1.

Confirming if SLEE Processes Are Running

You can use the check utility to confirm that the SLEE started up correctly and that all SLEE processes are running by entering the following command:

```
check -av
```

Getting Parsing Information on SLEE Processes

You can use the check utility to get parsing information about SLEE processes by entering the following command:

```
check -a[mode] [-j] [-s configuration_file]
```

where *mode* is one of:

Mode	Description
a	Show parsed applications.
A	Show SLEE view of applications.
i	Show parsed interfaces.
I	Show SLEE view of interfaces
w	Show parsed watchdog information.
s	Show parsed service information.
P	Show parsed parameter information.
k	Show parsed service key information.
e	Show parsed event information.
c	Show all parsed information.
v	Verify that parsed applications and interfaces are running.

By default, check connects to the SLEE and you use the **-a** command-line parameter to parse the default **/IN/service_packages/SLEE/etc/SLEE.cfg** file. To find out which SLEE processes should be available by parsing the **SLEE.cfg** file without connecting to the SLEE, use the **-j** command-line parameter; for example:

```
check -ai -j
```

You can then check which SLEE interfaces are running by using the mode **l** with the **-a** command-line parameter; for example:

```
 check -al
```

You can parse a local configuration file instead of the default **/IN/service_packages/SLEE/etc/SLEE.cfg** file by using the **-s** command-line parameter; for example:

```
check -a[mode] -s configuration_file
```

where *configuration_file* is the name of the local configuration file being parsed.

Checking Correlation IDs

You can use the check utility to get information about correlation IDs. To perform a correlation ID check, enter the following command:

```
check -g [-B] delay [count]
```

where:

Command Option	Description
-g	Continuous output, with a header.
-B	Do not print the report header.
delay	The number of seconds before the specified report is repeated. *delay* must be greater than 0 (zero).
count	The number of times the report should be repeated. *count* must be greater than 0 (zero).

Testing Event Consumption, Spillage, and Leakage

You can use the check utility to test event consumption and event spill-over and to test for event leakage.

Note: You are recommended to use this mode of operation only on test systems. Events that are either accidentally or intentionally leaked will not be released until the SLEE is restarted; this can result in system instability. Therefore, do not to use this mode of operation on a production system.

You test consumed events by running two check sessions concurrently; one to display information about event consumption and the other to monitor event usage.

To monitor event usage, run check in batch mode. See *Checking the SLEE by Using Batch Mode* (on page 29) for more information.

To test consumed events, enter the following command:

```
check -xevent_size -yevent_number [-rinterval]
```

where:

Command Option	Description
-x*event_size*	Sets the size of events to consume, where *event_size* specifies the size of events in bytes.
-y*event_number*	Sets the number of events of *event_size* to consume, where *event_number* specifies the number of events.
-r*interval*	Displays events consumed or deleted at intervals specified by *interval*, where *interval* can be greater than `0` (zero). For example, if 1000 events are consumed and *interval* is set to 100, then instead of displaying 1000 events in the check report, only every 100th event is reported. Do not display events consumed or deleted if *interval* is `0`.

After reporting on consumed events, the check utility displays the check menu options. You can then select to:

- Consume another `y` events
- Free all events consumed in this session
- Confirm or abandon consumption of leaked events

See *Main Menu Options* (on page 32) for more information.

Main Menu Options

To access the main menu options from the check report menu, run check without specifying any command-line options; for example, enter the following command:

```
check
```

The following menu displays:

```
Check which type of object?

    1: Dialogs
    2: Events
    3: Calls
    4: Services
    5: Applications
    6: Application Instances
    7: Interface Instances
    8: General Status
    9: Free Objects
    a: Event Queues
    b: Repeat consume events
    c: Free consumed events
    d: Leak consumed events
    e: Clear maximum event usage and maximum over spill counters
    f: Correlation Ids
    q: Quit
Select:
```

To select an option, type the character before the colon for the option you want.

The following table describes the check report menu options:

Option	Description
1: Dialogs	Reports the number of open SLEE dialogs.
2: Events	Reports the number of current SLEE events.
3: Calls	Reports the number of current calls.
4: Services	Reports the number of SLEE services that are running.
5: Applications	Reports the number of SLEE applications that are currently running.
6: Application Instances	Reports the number of instances of SLEE applications that are currently running.
7: Interface Instances	Reports the number of SLEE interface instances that are currently running.
8:General Status	Reports general information and statistics about the SLEE.
9: Free Objects	Reports the number of free objects still available in the SLEE shared memory.
a: Event Queues	Reports the number of events queued and waiting for the application or interface to process.
b: Repeat consume events	Consumes the next *event_number* events, where *event_number* is the number of events consumed when you enter the following syntax to check event size and event type: `check -xevent_size -yevent_number -rinterval`
c: Free consumed events	Frees all events consumed in this session.
d: Leak consumed events	Allows for the following menu extensions: c: Confirm leak consumed events d: Abandon leak
e: Clear maximum event usage and maximum over spill counters	Clears data logged on maximum event usage and maximum over-spill counters.
f: Correlation Ids	Reports correlation ID states.
q: Quit	Exits from check.

Output

check writes reports to stdout. The different reports have different formats. Reports will be the same whether run from the command line or from the check report menu.

Note: The total number of events reported by check will not match the total event number specified in the configuration file. This is because SLEE reserves a set of events for exclusive use by the watchdog process.

check writes error messages to the system messages file.

General reports

If you choose option 8 from the check report menu, check outputs a summary report of the information available from the other reports.

Example:

The following report is an example of a general report.

```
Select:  8
SLEE Status Report


Service:        ACS
Using application:       0xc0013d28


Service:        ACS_Outgoing
Using application:       0xc0013d28


Application:    slee_acs at 0xc0013d28
Contains the following Instances....
Instance at :   0xc0014b88
Process ID:     5493
Status: 3
Call Count:     0


Free Dialogs:   70000
Free Applications:      9
Free Application Instances:     90
Free Services:  8
Free Events:    49998
Free Calls:     25000
```

Exiting check

To exit from the check report running in periodic report mode, press **Ctrl**+**C**.

To exit from the menu, choose option `q`.

sfVerify

Purpose

sfVerify creates the **statistics.bin** file that is needed to run statsIF.

Configuration

sfVerify supports the following command-line options:

Usage:

```
sfVerify [-v--verbose-c--commit-f--force] [-d path --dir path] [-o filename --output
filename] [-s KB --size KB]
```

The available parameters are:

Parameter	Default	Description
`-v` `--verbose`	false	Controls the amount of information output from the program.
`-c` `--commit`	false	Commits the changes to the stats interface. This will only work if the SLEE is running and the stats interface is configured. This parameter writes the output file and then signals the stats interface to reread the configuration file statistics.bin.
`-f` `--force`	false	Force the commit without asking the user.
`-d` `--dir`	stats_defn	The directory to import the statistics definitions from.
`-o` `--output`	output.defn	The name of the output file.
`-s` `--size`	128	The maximum size in Kb for the stats interface output files.

Note: Any of the parameters (except the period) can either be a single word, or specified as a quote-delimited string.

Import files

The import files for the statistics interface take the following form:

```
applicationName statisticName description period comment
```

Output

The output file used by the statistics interface is:

```
/IN/service_packages/SLEE/etc/statistics.bin
```

sfVerify writes error messages to the system messages file.

Chapter 4

Background Processes

Overview

Introduction

This chapter explains the processes that are started automatically by Service Logic Execution Environment (SLEE).

Note: This chapter also includes some plug-ins to background processes which do not run independently.

In this chapter

This chapter contains the following topics.

replicationIF

Purpose

replicationIF responds to SLEE replication events by sending data to another machine (usually the SMS).

Startup

replicationIF is started by the following line in **SLEE.cfg**:

```
INTERFACE=replicationIF replicationIF.sh /IN/service_packages/SLEE/bin 1 EVENT
```

For more information about using the INTERFACE entry, see *INTERFACE* (on page 17).

Configuration

replicationIF supports the following command line parameters:

```
replicationIF -r node -d microsecs -enableNonBlockWrite 0|1 -infoResetInterval secs
-infoReportInterval secs -updateRequestAcksPendingAlarmThreshold threshold -
updateRequestsPendingAlarmThreshold num -updateRequestResendLimit microsecs -
writeBlockSleepTime microsecs -writeBlockPendingUpdateLimit num
```

`-r`

Syntax:	`-r` *node*
Description:	The node number of the requester node (that is, the node number of the replicationIF itself).
Type:	Integer
Optionality:	Mandatory (disallowed default of 0 used if not set).
Allowed:	A number between 512 and 1023.

Default:	0
Notes:	
Example:	`-r 601`

-d

Syntax:	`-d` *`microsecs`*
Description:	The number of microseconds between processing large SLEE events.
Type:	Integer
Optionality:	Optional (default used if not set).
Allowed:	
Default:	1000
Notes:	
Example:	

-enableNonBlockWrite

Syntax:	`-enableNonBlockWrite` *`0|1`*
Description:	Enables or disables non-blocking writes.
Type:	Boolean
Optionality:	Optional (default used if not set)
Allowed:	• 0 – Disabled. replicationIF can block writes during congestion. Generally, this blocking state is interrupted by a signal generated when data arrives from smsMaster. • 1 – Enabled. replicationIF does not block writes.
Default:	0 (disabled)
Notes:	
Example:	`-enableNonBlockWrite 1`

-infoResetInterval

Syntax:	`-infoResetInterval` *`secs`*
Description:	The minimum amount of time, in seconds, before update requester information/statistics are reset. No reset occurs when the following additional conditions are true: • A resend is in progress after a connection loss (or a write fail) • A transmit blocking/congestion state exists • Update requests are yet to be acknowledged
Type:	Integer
Optionality:	Optional (default used if not set)
Allowed:	An integer greater than 0
Default:	300
Notes:	
Example:	`-infoResetInterval 500`

-infoReportInterval

Syntax:	`-infoReportInterval` *`secs`*
Description:	The minimum interval, in seconds, between updates to requester information/statistics.
Type:	Integer

Optionality:	Optional (default used if not set)
Allowed:	A number greater than 0
Default:	10
Notes:	If you set this parameter, you must also enable the REP_UPDATE debug flag.
Example:	`-infoReportInterval 5`

-updateRequestAcksPendingAlarmThreshold

Syntax:	`-updateRequestAcksPendingAlarmThreshold` *threshold*
Description:	The threshold for the number of outstanding update requests requiring an acknowledgment from smsMaster before an alarm is generated.
Type:	Integer
Optionality:	Optional (default used if not set)
Allowed:	An integer greater than 5,000
Default:	40,000
Notes:	
Example:	`-updateRequestAcksPendingAlarmTheshold 10000`

-updateRequestsPendingAlarmThreshold

Syntax:	`-updateRequestsPendingAlarmTheshold` *num*
Description:	The threshold for the number of pending update requests to be sent to smsMaster before an alarm is generated.
Type:	Integer
Optionality:	Optional (default used if not set)
Allowed:	An integer greater than 1,000
Default:	20,000
Notes:	
Example:	`-updateRequestsPendingAlarmThreshold 10000`

-updateRequestResendLimit

Syntax:	`-updateRequestResendLimit` *microsec*
Description:	After connection loss, all unacknowledged update requests are re-transmitted. This can result in a large number of update requests, depending on the rate of acknowledgments from smsMaster. `updateRequestResendLimit` specifies the duration of the re-transmit period, in microseconds, with no handling of new update requests. After this duration, update requests are re-transmitted in maximal block sizes determined by `writeBlockPendingUpdateLimit`. This allows a large re-transmit and the ability to service newly arriving update requests.
Type:	Integer
Optionality:	Optional (default used if not set)
Allowed:	An integer from 300 to 20,0000 (inclusive)
Default:	500
Notes:	
Example:	`-updateRequestResendLimit 1000`

-writeBlockSleepTime

Syntax:	`-writeBlockSleepTime` *`microsecs`*
Description:	The latency, in microseconds, between attempts to write a further update. This allows time for smsMaster to consume data.
Type:	Integer
Optionality:	Optional (default used if not set)
Allowed:	A number greater than 0
Default:	10,000
Notes:	
Example:	`-writeBlockSleepTime 9000`

-writeBlockPendingUpdateLimit

Syntax:	`-writeBlockPendingUpdateLimit` *`num`*
Description:	The maximum number of latencies associated with sending large amounts of pending updates. This parameter helps with transmission congestion when update requests accumulate and are then transmitted before new updates. These writes are limited to blocks of size `writeBlockPendingUpdateLimit`.
Type:	Integer
Optionality:	Optional (default used if not set)
Allowed:	An integer greater than 0
Default:	10
Notes:	
Example:	`-writeBlockPendingUpdateLimit 5`

tcRelayApp

Purpose

tcRelayApp is a SLEE application which relays TCAP primitives.   This enables SLEE interfaces to send TCAP primitives to other SLEE interfaces (particularly TCAP IF).

You can use tcRelayApp to add destination number and originating number information to the TCAP primitive.

Startup

tcRelayApp is started by the following line in SLEE.cfg:

```
APPLICATION=tcRelayApp tcRelayApp.sh /IN/service_packages/SLEE/bin 1 1
```

Configuration

tcRelayApp supports the following parameters in each line of tcRelayMappings.def:

`serviceHandle IF name dest ssn dest_pc dest_GT orig_ssn orig_pc`

The available parameters are:

Parameters	Default	Description
serviceHandle		Incoming service handler. The SLEE service handle this message has been sent to. (Required.)

Parameters	Default	Description
IF name		Outgoing interface name. The SLEE interface name the message should be forwarded to. (Required.)
dest ssn		Destination SSN. The Destination SSN value which the TCAP primitive should have. (Required.)
dest pc		Destination PC. The Destination Point Code value which the TCAP primitive should have. (Required.)
dest gt		Destination GT. The Destination Global Title value which the TCAP primitive should have. (Optional, use "-" to not specify.)
orig ssn		Originating SSN. The Originating SSN value which the TCAP primitive should have. (Optional.)
orig pc		Originating PC. The Originating PC value which the TCAP primitive should have. (Optional.)

Failure

If tcRelayApp fails, TCAP primitives which must be relayed through the SLEE will fail. If the process is not running, the SLEE watchdog will attempt to restart the service.

Output

tcRelayApp writes error messages to the system messages file, and also writes additional output to:

```
/IN/service_packages/SLEE/tmp/tcRelayApp.log
```

watchdog

Purpose

The watchdog process checks on SLEE processes to see that they are running as expected. It also ensures that the current time of day is correct in the shared memory segment.

The watchdog process performs the following actions:

- Every 10 milliseconds the watchdog writes the current time of day to the shared memory segment. If it is unsuccessful, it writes an error code to the shared memory segment so that SLEE processes can respond appropriately.
- When a process starts, the watchdog begins a throttle period, which is the product of WATCHDOGCYCLETIME times the value of WATCHDOGTHROTTLECYCLES. If the process dies or becomes unresponsive during the throttle period, the watchdog restarts it after an additional watchdog cycle (WATCHDOGCYCLETIME). If the process dies or becomes unresponsive *after* the throttle period, the watchdog process treats it in the same way as other unresponsive processes.

Note: The throttle period prevents the watchdog from continuously restarting a process that dies immediately, for example due to an incorrect configuration.

- Every 100 milliseconds the watchdog scans SLEE processes to see if they are still processing and responsive. If a process is not processing at all, watchdog considers it a zombie and restarts it immediately. If a process is running but has not responded to a watchdog message within a watchdog cycle the watchdog considers that process unresponsive and marks it as suspect. If a suspect cycle responds to a watchdog message within one watchdog cycle, the watchdog reverses its status from suspect to normal.

- If a suspect process remains unresponsive, the watchdog process allows it to continue processing for a period of time that is the product of WATCHDOGSUSPECTCYCLES times the value of WATCHDOGCYCLETIME. At that point, the watchdog process terminates the unresponsive process and restarts it.

Note: The WATCHDOGSUSPECTCYCLES parameter is designed to allow a process sufficient time to finish dumping core for diagnostic purposes. It prevents an immediate restart that could result in a truncated core file, which would be useless.

You can override WATCHDOGSUSPECTCYCLES and WATCHDOGTHROTTLECYCLES for a given application or interface. See the *APPLICATION* (on page 23) and *INTERFACE* (on page 17) configuration parameters for more information.

The watchdog process also tracks SLEE resource usage. If a resource drops below 80% of the start value, the watchdog alerts the user to the condition. A notice is posted when the value rises back up to 70% of the start value. The watchdog keeps track of usage for the following resources: dialogs, call instances, and event lists.

When the watchdog begins a check loop, it starts a timer running to ensure that it will not remain deadlocked on a semaphore forever. If the timer expires the watchdog restarts the SLEE.

Startup

The watchdog application resides in the following shared library:

```
/IN/service_packages/SLEE/lib/libSleeWatchdog.so
```

It is started by the following line in SLEE.cfg:

```
APPLICATION=tcRelayApp tcRelayApp.sh /IN/service_packages/SLEE/bin 1 1
```

Configuration

The watchdog process accepts the following parameter options from the **SLEE.cfg** file. The watchdog parameters are set at installation and should not need changing. They define the location and cycle time for the watchdog.

```
WATCHDOG=/IN/service_packages/SLEE/bin/ watchdog
WATCHDOGCYCLETIME=30
WATCHDOGSUSPECTCYCLES = 1
WATCHDOGTHROTTLECYCLES = 1
```

The available parameters are:

Parameters	Default	Description
`WATCHDOG`		The path and binary filename for the watchdog executable.
`WATCHDOGCYCLETIME`	30	The number of seconds between checks on SUSPECT processes. For more information about SUSPECT processes, see Purpose.
`WATCHDOGSUSPECTCYCLES`	1	Specifies the number of watchdog cycles to wait before restarting a non-responsive process.The delay allows an unresponsive process time to finish dumping core for diagnostic purposes before it is restarted.

Parameters	Default	Description
`WATCHDOGTHROTTLECYCLES`	1	Specifies the number of watchdog cycles to use for the throttle period (WATCHDOGTHROTTLECYCLES times WATCHDOGCYCLETIME), which begins when an application starts. If an application or interface dies or becomes unresponsive *during* the throttle period, the watchdog process restarts it after an additional watchdog cycle. This prevents the watchdog process from continuously restarting processes that die immediately due to incorrect configuration, for example. If the application or interface dies or becomes unresponsive *after* the throttle period, the watchdog process treats it like any other unresponsive process.

Failure

If the watchdog fails, SLEE processes will not be monitored. This will mean SLEE processes are not restarted if they fail. You must restart the SLEE. For more information about restarting the SLEE, see *Tools and Utilities* (on page 27).

Output

watchdog writes error messages to the system messages file.

Chapter 5

Troubleshooting

Overview

Introduction

This chapter explains the important processes on each of the server components in NCC, 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 NCC 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*.

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.

SLEE failing on startup

This table describes possible reasons why the SLEE may be failing to startup:

Reason	Remedy	Alarms
sleeStartup could not parse the **SLEE.cfg** file.	Check that the **SLEE.cfg** file exists in the expected location and that it can be read. Check that the syntax of the file is correct. For more information about the **SLEE.cfg** file, see *Configuration* (on page 9).	

Chapter 6

About Installation and Removal

Overview

Introduction

This chapter provides information about the installed components for the Oracle Communications Network Charging and Control (NCC) 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*:

- NCC system requirements
- Pre-installation tasks
- Installing and removing NCC packages

SLEE packages

An installation of SLEE includes the following packages, on the:

- SLC:
  - SLEE
- VWS:
  - SLEE

Checking the Installation

Introduction

Refer to this checklist to ensure that SLEE has installed correctly.

Checklist

Follow these steps in this checklist to ensure the SLEE has been installed on the SLC machine correctly.

Step	Action
1	Log onto the machine as root.
2	Check the following directory structure exists with subdirectories: • **/IN/service_packages/SLEE**
3	Check the directories and subdirectories are all owned by: root user (group other)