Oracle® Communications Network Charging and Control

System Administrator's Guide

Release 15.2

January 2026

Copyright

Contents

About This Document

Scope

The scope of this document includes all functionality a user must know in order to effectively operate the Oracle Communications Network Charging and Control (NCC) application. It does not include detailed design of the service.

Audience

This guide is written primarily for NCC administrators. However, the overview sections of the document are useful to anyone requiring an introduction.

Prerequisites

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.

Although it is not a prerequisite to using this guide, familiarity with the target platform would be an advantage.

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:

- *Installation Guide*
- *Configuration User's Guide*

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.

Chapter 1

NCC System Architecture

Overview

Introduction

This chapter introduces the Oracle Communications Network Charging and Control (NCC) system architecture.

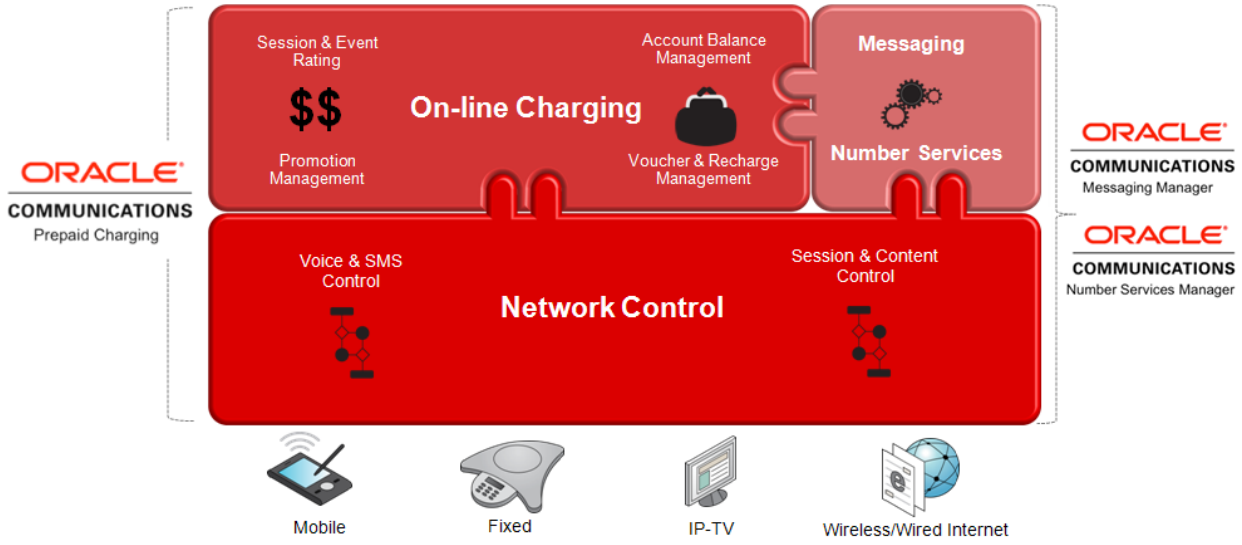
In this chapter

This chapter contains the following topics.

NCC System Architecture Overview

Architecture diagram

This diagram depicts the NCC system from a network architecture perspective:



System components

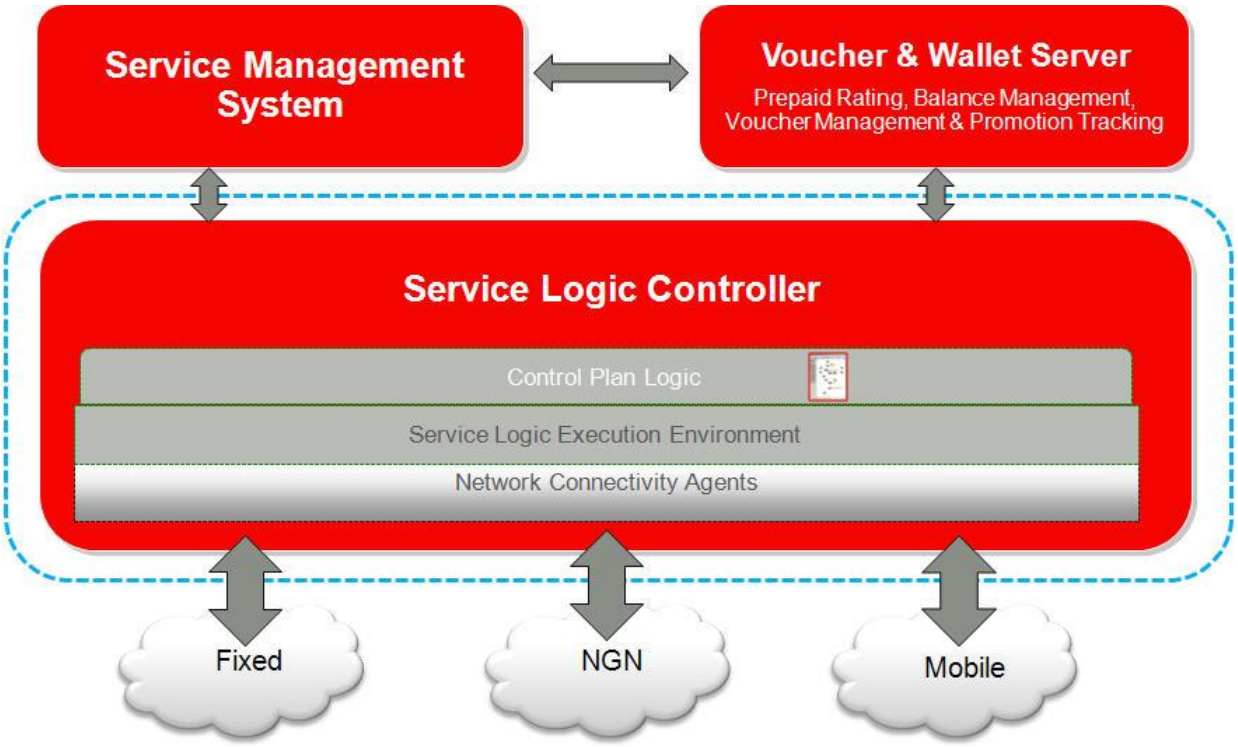
The architect diagram, from the bottom layer up, we see:

- A number of disparate telecommunications services (for example, mobile, fixed, IP) showing the ability of the NCC system to handle convergence.
- Secondly, the network control aspect is handled using services templates (for example, fixed, mobile, data and TV).

- Finally the on-line charging layer handles the service rating, subscriber balance management (and voucher/promotion management if appropriate).
- Two optional additional products are also depicted in this diagram, **Messaging Manager** and **Number Services Manager**, which are out of the scope of this document.

Server components diagram

This diagram shows how the three main server components of the NCC system combine to form the system architecture.



Server descriptions

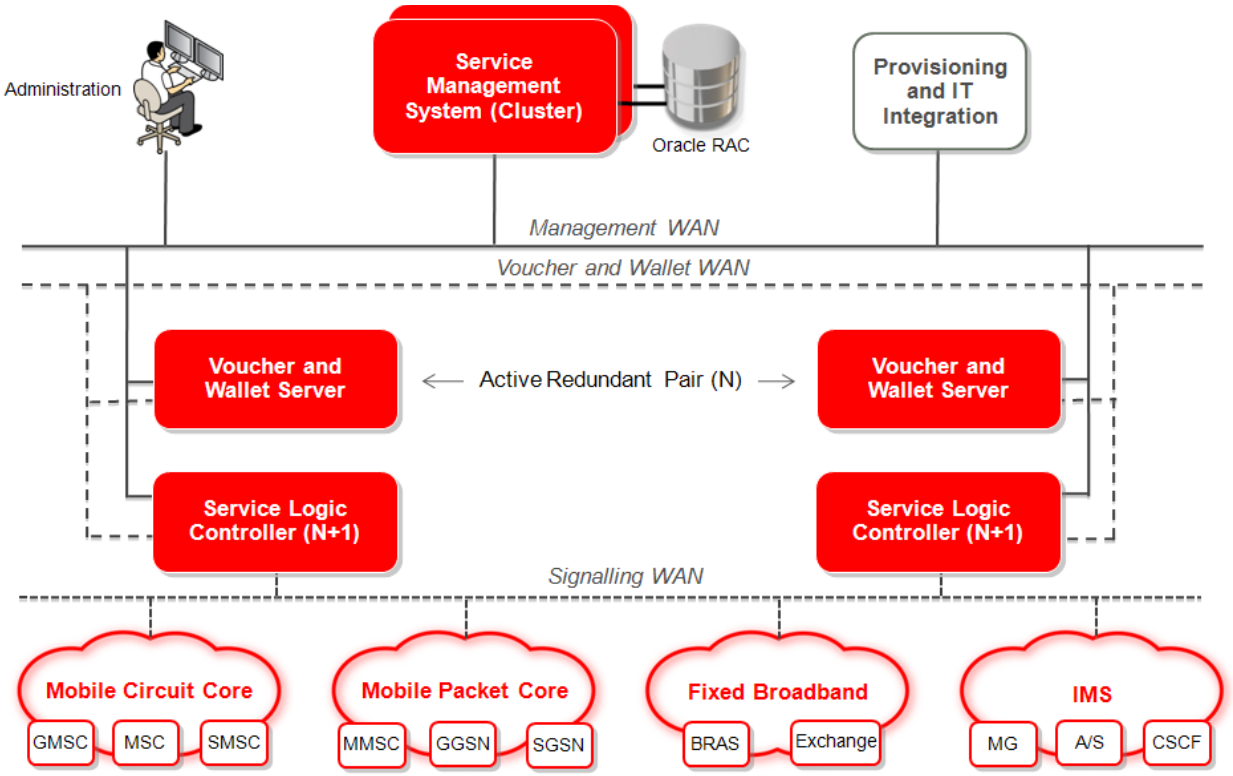
The server components are as follows:

- The Service Logic Controller (SLC) is the main interface to the network and handles all service processing (voice/SMS/data content).
  Service processing is handled through the Service Logic Execution Environment (SLEE), with the various network connectivity agents (for example, diameter, radius, MAP, SIP and the ACS control plan logic, defining the service logic for all enabled services.
- The Service Management System (SMS) provides the base system management functionality, including:
  - The Java administration UI
  - Centralized data storage
  - Replication functionality
- The Voucher and Wallet Server (VWS) is essentially the billing component of the system (this could also be provided by a third-party billing system, such as Oracle Communications Billing and Revenue Management (BRM).
  Billing provides:
  - Prepaid Rating
  - Balance Management
  - Voucher Management

- Promotion Tracking facilities

Multiple servers configuration diagram

The system architecture can support multiple server configurations as shown in this diagram:

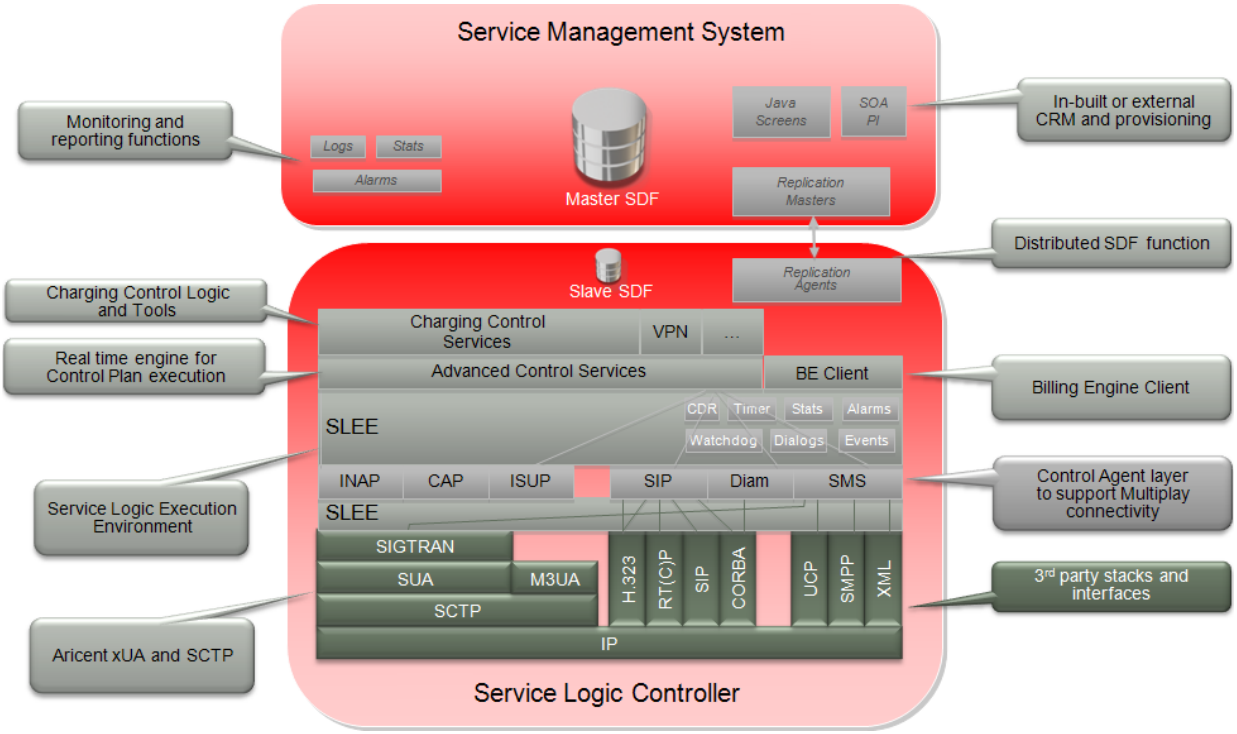


Note: The configuration of the VWS in a redundant pair set-up and the SLC running in an N+1 configuration. Here the SMS is set up in a redundant cluster configuration using Oracle RAC for the database component.

SMS and SLC Server Operation

Operation diagram

This diagram shows the main components and operation of the SMS and SLC servers:



SLC component list

This table describes the main components for the Service Logic Controller.

Component	Description
Charging Control Services (CCS)	Provides the charging control logic and tools.
Advanced Control Services (ACS)	Provides the real time engine for control plan execution, effectively the call processing engine.
Billing Engine Client	Provides the interface which processes requests from the call processing engine to the Voucher and Wallet servers.
Service Logic Execution Environment (SLEE)	Routes calls to the ACS and to other machines through the SLEE interfaces (TCAP and Billing Engine Client).

SMS component list

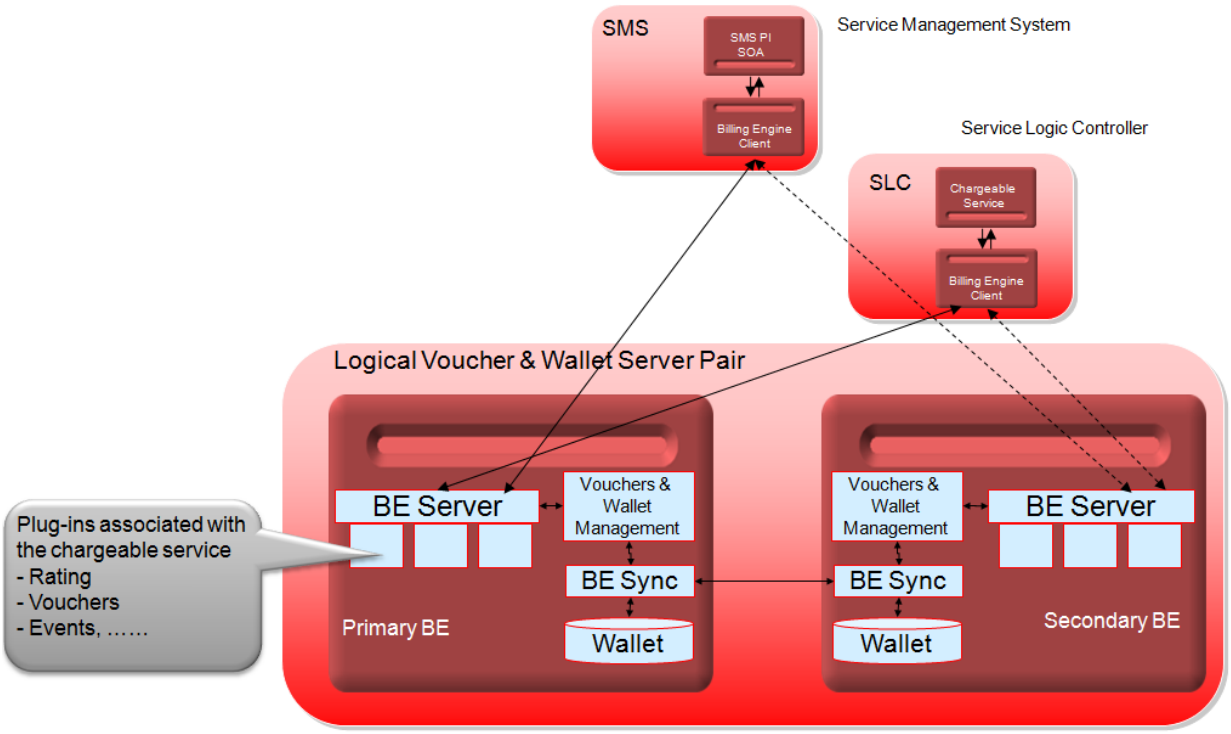
This list describes the main components for the Service Management System

- Centralized storage of logs, alarms, and statistics.
- In-built CRM system which can be provisioned directly or externally via the provisioning interface.
- Replication method used to transfer relevant data from the main database on the SMS to the VWS and SLC servers, including:
  - Subscriber and account wallet data
  - Tariff and rate tables, for example.

VWS Server Operation

Operation diagram

This diagram shows the main components and operation of the Voucher and Wallet (VWS) server:



VWS component list

This list describes the main components for the Voucher and Wallet Server.

- The BE Sync component, which runs on each VWS, synchronizes the subscriber wallet data between the databases on the two servers, here depicted running in a redundant pair set-up.
- The BE Server component, which runs on each VWS, handles all incoming requests from the SMS and SLC client processes, and can be extended using plug-ins.

Chapter 2

Service Management and Control

Overview

Introduction

This chapter explains the management and control of the NCC product.

In this chapter

This chapter contains the following topics.

Service Management and Control Overview

Introduction

The NCC solution is a group of programs, or applications that runs on both Oracle Linux and database.

A familiarity with Oracle Linux and Oracle database concepts and commands is necessary to fully understand the management and control of the applications that make up the whole solution.

NCC concepts

A concept to think about when considering service management is solution redundancy.

The NCC solution uses both the **N+1** and **2N** concepts to increase service reliability and greatly reducing the chances of a complete Service outage. These redundancy approaches are used in the following ways:

SLC usage

The SLC (an SCP in signalling terms) nodes handle network traffic in an **N+1** configuration.

All nodes independently handle traffic at the same time. In the case of node failure, or service interruption, the remaining node(s), have the extra capacity to handle the increased load of the unavailable node (up to the projected peak traffic load).

VWS usage

The VWS nodes work in the more traditional **2N**, or hot standby, configuration. This consists of a logical pair of servers running in a primary and secondary node configuration that are constantly updating and synchronizing themselves.

In the case of node failure or service interruption to the primary node, the secondary node provides instant and uninterrupted backup until the primary node is back in service.

SMS usage

The default installation of SMS node does not provide high availability. For more information about high availability, see *NCC High Availability Operations Guide*.

Management and control methods

There are two main methods of service management and control of NCC components:

- Service managed applications
  - Daemon is a core UNIX process (PID=1) that spawns all other processes. The **/etc/inittab** file controls the init daemon and can be used to manage running applications.
- SLEE managed applications

  SLEE is a computing term and stands for Service Logic Execution Environment. The NCC SLEE manages a group of applications that can communicate with each other while efficiently sharing resources.

Note:

- All the NCC servers have some components that are managed by the init daemon.
- Only the SLC and VWS servers have components that are managed by the SLEE.

init Daemon Management

init daemon process

The init daemon process "is the default primordial user process" on a UNIX system (see `$ man init` information).

Backwards compatibility

For backwards compatibility, init also starts and restarts general processes according to rules specified in the **/etc/inittab** file and the start/stop scripts defined in the legacy **/etc/init.d** and **/etc/rc?.d** directories (see `$ man` page for `inittab` and `init.d`).

inittab file

NCC uses the inittab file to manage and control a number of its daemon processes. All NCC init managed processes are configured during the installation process to be run when the server is in run-level 3 or 4 (rstate).

```
$ man inittab

 ...skipped...
The inittab file is composed of entries that are position dependent and have the
following format:
id:rstate:action:process
```

NCC process identification

The easiest way to identify the NCC process which are managed by init is to search (`grep` command) for the "IN" string in the **/etc/inittab** file, which will list the process startup scripts.

```
$ grep "IN" /etc/inittab

sms1:34:respawn:su - smf_oper -c "exec /IN/service_packages/SMS/bin/smsAlarmRelayStartup.sh >>
/IN/service_packages/SMS/tmp/smsAlarmRelay.log 2>&1" > /dev/null 2>&1 0<&1
sms2:34:respawn:su - smf_oper -c "exec /IN/service_packages/SMS/bin/smsNamingServerStartup.sh >>
/IN/service_packages/SMS/tmp/smsNamingServer.log 2>&1" > /dev/null 2>&1 0<&1
```

```
...skipped...
```

Tip: The general naming convention format of the process startup scripts is the name of the running process with "Startup.sh" on the end:

Format: *process_name***Startup.sh**

Example: **smsAlarmRelayStartup.sh**

Process running checking

To check if the process is running you use the UNIX **ps** command (process status – see `$ man ps` information) and search for the process name.

```
$ ps -ef | grep smsAlarmRelay
smf_oper   941     1   0   Oct 04 ?          55:53 /IN/service_packages/SMS/bin/smsAlarmRelay -e
-u /
smf_oper  3554  3205   0 22:04:09 pts/1       0:00 grep smsAlarmRelay
$
```

It is worthwhile becoming familiar with the NCC processes managed by init.

Stop and Start Processes

Configuring Service Daemons

Restart the NCC service daemons in all the nodes by running the following command:

```
init q
```

or use kill commands to kill the NCC service daemon

Changing the run level

At times it may be necessary to stop all the NCC init managed processes at once. You can manually edit the inittab file and comment out all the NCC processes but this can be fiddly, takes time and can be prone to error.

The quickest and easiest way to stop all the processes is to simply lower the run-level of the server to **state 2 (multi-user)**. This instructs the init daemon to stop any of its managed processes, specified in the inittab file, that are not configured to run in this level (the rstate field).

The start and stop scripts in the **/etc/rc***rstate***.d** directories will also run.

As mentioned earlier the NCC processes are configured to run in either states 3 or 4 so that when the `init 2` command is run (instructs the init daemon to move into run-level 2), then init will automatically terminate all its managed processes that are not configured to run in this new level.

To automatically restart these stopped processes again, you move back into run-level 3 with the `init 3` command.

You must be super-user to change the server run-level.

Generally speaking you would only ever want to do use this method during platform maintenance, be it either; application, database, or server related interventions.

Example - updateLoader

Follow these steps to stop and start the updateLoader process.

Step	Action

1	Become the root user, enter:
	`$ su – root`
	`Password: *******`
	`Sourcing /etc/profile.ORA`

2	Check the updateLoader process is running, enter:
	`ps -ef |grep update`
	`smf_oper   850     1   0   Oct 04 ?           0:22`
	`/IN/service_packages/SMS/bin/updateLoader -nodeid 301`
	`    root 19354 19349   0 22:34:08 pts/1       0:00 grep update`

3	Edit (use your preferred text editor) the **/etc/inittab** file and comment out (add the # character to beginning of line) the **updateLoaderStartup.sh** entry and save the change.
	`vi /etc/inittab`
	`...skipped...`
	`#scp5:34:respawn:su - smf_oper -c "exec`
	`/IN/service_packages/SMS/bin/updateLoaderStartup.sh >>`
	`/IN/service_packages/SMS/tmp/updateLoader.log 2>&1" > /dev/null 2>&1 0<&1`
	`[Esc]`
	`:wq`
	`"/etc/inittab" 36 lines, 3045 characters`

4	Tell init to reread the inittab file: `init q`

5	Check the updateLoader process has stopped:
	`ps -ef | grep update`
	`    root 19431 19349   0 22:40:06 pts/1       0:00 grep update`

6	Edit the **/etc/inittab** file and remove the # from the **updateLoaderStartup.sh** entry and save the change:
	`vi /etc/inittab`
	`...skipped...`
	`scp5:34:respawn:su - smf_oper -c "exec`
	`/IN/service_packages/SMS/bin/updateLoaderStartup.sh >>`
	`/IN/service_packages/SMS/tmp/updateLoader.log 2>&1" > /dev/null 2>&1 0<&1`
	`[Esc]`
	`:wq`
	`"/etc/inittab" 36 lines, 3045 characters`

7	Tell init to reread the inittab file: `init q`

8	Check the updateLoader process has started and is running again:
	`ps -ef |grep update`
	`smf_oper 19475     1   0 22:42:43 ?           0:00`
	`/IN/service_packages/SMS/bin/updateLoader -nodeid 301`
	`    root 19566 19349   0 22:42:45 pts/1       0:00 grep update`

Example - smsMaster

Follow these steps to restart the smsMaster process.

Step	Action

1	Check the smsMaster process are running:
	`$ ps -ef | grep smsMaster`
	`smf_oper 978 1 0 Oct 04 ? 9:28 /IN/service_packages/SMS/bin/smsMaster -c`
	`smf_oper 1299 978 1 Oct 04 ? 653:19 /IN/service_packages/SMS/bin/smsMaster -c`
	`smf_oper 6138 3205 0 00:21:04 pts/1 0:00 grep smsMaster`
	Note: The second (Process ID or PID) and third columns (Parent PID or PPID) of the output. The PPID of 1, on the the first line, denotes it as the init process and the smsMaster process is a child process. The second smsMaster listed is child process of the first one as it's PPID matches the first lines PID.

2	As the smsMaster process owner (smf_oper) terminate the process with the kill command (signal the process to stop running, or exit): `$ kill 978` **Note:** As PID 978 is the parent process, the child process will inherit the SIGTERM signal (kill command's default signal (either; -TERM, or –15 switch)) from it's parent. It would be okay to specify both PIDS. Using the SIGTERM signal is best practice as it allows the child and parent processes to cleanup and close files before terminating itself. The super-user root has global ownership permissions allowing it to send signal to any process.

3	Check that the init daemon has instantly restarted the smsMaster process:
	`$ ps -ef | grep smsMaster`
	`smf_oper 6173 1 0 00:23:06 ? 0:01 /IN/service_packages/SMS/bin/smsMaster -c`
	`smf_oper 6188 6173 1 00:23:06 ? 0:24 /IN/service_packages/SMS/bin/smsMaster -c`
	`smf_oper 6800 3205 0 00:41:09 pts/1 0:00 grep smsMaster`
	Tip: If either of the previous smsMaster processes are still listed then this may indicate that the process is hung or stuck and may need a more forceful shutdown signal. If this is the case then the SIGKILL signal (either; -KILL or -9 switch) is recommended.
	`$ kill -9 978 1299`

SLEE Management

Introduction

The SLC and VWS servers have components that are managed by the NCC SLEE. Even though the services run on both these platforms are completely different, the concepts for how they are controlled are exactly the same.

In a nutshell, the SLEE provides a common environment for multiple different service logic applications to run in and simultaneously manage and communicate events between themselves in an orderly way, while interfacing with multiple external networks and/or applications.

SLEE control

Control of the SLEE is managed by a utility called SLEE Control and the command to use it is `slee-ctrl`.

The **slee-ctrl** tool provides the ability to stop, start, or restart the SLEE and, among other things, will verify that all SLEE processes were started, or have actually been stopped.

Basically it is a wrapper script that provides the user with extra checks and protection from causing issues. It still calls the SLEE start and stop scripts, all the while avoiding the pitfalls that inexperienced operators may have.

slee-ctrl simplifies SLEE control:

- Only one command name to remember.
- No need to remember which UNIX user you must be (it will remind you if it's wrong).
- One valid SLEE_USER must be configured and only this user or super-user root can run the start, stop, or restart commands.
- Do not need to remember the location or names of the SLEE start and stop scripts.
- Do not have to be in the **/IN/service_packages/SLEE/bin** directory to run the command.
- Verifies that all SLEE instance processes are terminated when the stop command issued.
- Verifies that no other SLEE instance processes are running when the start command issued.
- List the current process status of SLEE applications.
- Access to the SLEE check command which displays internal SLEE resource usage.
- Provides an audit trail by logging a history of when slee-ctrl commands were run.

slee-ctrl modes of operation

The slee-ctrl tool has two modes of use:

- Command line
- Interactive session mode

Documentation for slee-ctrl is in the form of a manual page. It is recommended that you read this to understand the full capabilities of the tool (enter **`man slee-ctrl`**). Also try entering **`slee-ctrl help`** on the command line for a full list of valid commands.

Example VWS start up

To start the VWS SLEE on the command line, enter:

```
$ su - ebe_oper
Password:
$ slee-ctrl start
```

```
SLEE Control: v,1.0.11: script v,1.21: functions v,1.48: pslist v,1.118

[ebe_oper] slee-ctrl> start
Info: slee_ctrl_lock_file: lock file created for "start" command.
Info: slee_ctrl_start: Verifying that no SLEE processes are running...
Info: slee_ctrl_start: 20101027-03:40:03 GMT: Running SLEE sleeStartup...
The SLEE will be up and running shortly. . . . . .
----------------------- Wed Oct 27 03:40:14 GMT 2010 -------------------------
 C APP  USER      PID PPID    STIME COMMAND
 1 SLEE ebe_oper 7739    1 03:40:04 /IN/service_packages/SLEE/bin/timerIF
16 SLEE ebe_oper 7755    1 03:40:04 /IN/service_packages/E2BE/bin/beVWARS
 1 SLEE ebe_oper 7756    1 03:40:04 /IN/service_packages/E2BE/bin/beSync
 1 SLEE ebe_oper 7769    1 03:40:04 /IN/service_packages/E2BE/bin/beServer
 4 SLEE ebe_oper 7778    1 03:40:05 /IN/service_packages/E2BE/bin/beGroveller
 1 SLEE ebe_oper 7781    1 03:40:05 /IN/service_packages/SLEE/bin/replicationIF
 1 SLEE ebe_oper 7785    1 03:40:05 /IN/service_packages/DAP/bin/dapIF
 1 SLEE ebe_oper 7786    1 03:40:05 N/service_packages/E2BE/bin/beEventStorageIF
```

```
 1 SLEE ebe_oper 7787    1 03:40:05 N/service_packages/E2BE/bin/beServiceTrigger
 1 SLEE ebe_oper 7788    1 03:40:05 service_packages/CCS/bin/ccsSLEEChangeDaemon
 1 SLEE ebe_oper 7789    1 03:40:05 /IN/service_packages/SLEE/bin//watchdog
total processes found = 29 [ 29 expected ]
================================ run-level 3 ================================
Info: slee_ctrl_lock_file: lock file deleted for "start" command.
```

Example SLC stop

To stop the SLC SLEE in session mode, enter:

```
$ su - acs_oper
Password:
$ slee-ctrl

SLEE Control: v,1.0.11: script v,1.21: functions v,1.48: pslist v,1.118

User acs_oper; session [16893]; terminal pts/2; started Wed Oct 27 03:41:57 GMT 2010

The following variables are set and will be used to run the SLEE.
 SLEE_USER    acs_oper
 SLEE_SCRIPT  /IN/service_packages/SLEE/bin/slee.sh
 SLEE_CONFIG  /IN/service_packages/SLEE/etc/SLEE.cfg
 SLEE_LOG     /IN/service_packages/SLEE/tmp/SLEE.log

Type help at prompt for valid commands.

 [acs_oper] slee-ctrl> stop
 Type "yes" if you are sure: yes
Info: slee_ctrl_lock_file: lock file created for "stop" command.
Info: slee_ctrl_stop: 20101027-03:42:40 GMT: Running SLEE stop.sh...
Oct 27 03:42:40 stop(17106) SleeRoot::shutdown()=9 sleeRoot.cc@883: SLEE Shutdown
SLEE: Using shared memory offset: 0xc0000000
SleeRoot: Shutting Down...
SleeRoot: SIGUSR1 Watchdog (PID 14290).
SleeRoot: Disable all services...
SleeRoot: ...done
SleeRoot: Send management end to all interfaces...
SleeRoot: ...done
SleeRoot: Send management end to all applications...
SleeRoot: ...done
SleeRoot: Wait to allow service/application completion...
SleeRoot: ...done
SleeRoot: Kill all interfaces...
SleeRoot: SIGKILL Interface (PID 14262).
...skipped...
SleeRoot: SIGKILL Interface (PID 14289).
SleeRoot: ...done
SleeRoot: Kill all applications...
SleeRoot: SIGKILL Application (PID 14251).
...skipped...
SleeRoot: SIGKILL Application (PID 14261).
SleeRoot: ...done
SleeRoot: Delete semaphore manager.
SleeRoot: Delete shared memory.
SleeRoot: All Done.
Info: slee_ctrl_clean: cleaning SLEE shared memory and semaphores...
Info: slee_verify_stop: Verifying all SLEE process(es) have stopped...
Info: slee_verify_stop: No running SLEE processes found.
Info: slee_verify_stop: /IN/service_packages/ACS/.slee_file deleted.
Info: slee_ctrl_lock_file: lock file deleted for "stop" command.
[acs_oper] slee-ctrl> quit
 $
```

Note: In **session mode** you are prompted to type in yes if you want to stop or restart the SLEE. This safety check is not enforced when run from the command line.

Example VWS smf_oper restart

To restart the VWS SLEE as the smf_oper user, enter:

```
$ slee-ctrl restart

SLEE Control: v,1.0.11: script v,1.21: functions v,1.48: pslist v,1.118

[smf_oper] slee-ctrl> restart
Error: slee_ctrl_confirm_user: You must be SLEE_USER [ ebe_oper ], or super user, to run
"restart" command.
$
```

Example stopped processes check

To check the status of the SLEE stopped processes, enter:

[acs_oper] **slee-ctrl> status**

```
----------------------- Wed Oct 27 03:53:26 GMT 2010 -------------------------
C APP  USER       PID PPID    STIME COMMAND
6 SLEE acs_oper 17509    1 03:51:20 /IN/service_packages/ACS/bin/slee_acs
1 SLEE acs_oper 17511    1 03:51:20 /IN/service_packages/SLEE/bin/capgw
1 SLEE acs_oper 17512    1 03:51:20 /IN/service_packages/RAP/bin/rap
1 SLEE acs_oper 17513    1 03:51:20 /IN/service_packages/LCP/bin/locApp
1 SLEE acs_oper 17514    1 03:51:20 /IN/service_packages/ACSUSC/bin/slee_acs
1 SLEE acs_oper 17515    1 03:51:20 /IN/service_packages/UIS/bin/ussdgw
1 SLEE acs_oper 17516    1 03:51:20 /IN/service_packages/SLEE/bin/timerIF
1 SLEE acs_oper 17517    1 03:51:20 /IN/service_packages/SLEE/bin/alarmIF
1 SLEE acs_oper 17518    1 03:51:20 N/service_packages/ACS/bin/acsStatsLocalSLEE
1 SLEE acs_oper 17519    1 03:51:20 /IN/service_packages/SLEE/bin/replicationIF
1 SLEE acs_oper 17520    1 03:51:20 /IN/service_packages/E2BE/bin/BeClient
1 SLEE acs_oper 17525    1 03:51:20 /IN/service_packages/OSD/bin/osdInterface
1 SLEE acs_oper 17529    1 03:51:21 /IN/service_packages/SCA/bin/sca
1 SLEE acs_oper 17533    1 03:51:21 /IN/service_packages/RIMS/bin/rims
1 SLEE acs_oper 17536    1 03:51:21 /IN/service_packages/XMS/bin/xmsTrigger
4 SLEE acs_oper 17537    1 03:51:21 /IN/service_packages/SLEE/bin/m3ua_if
1 SLEE acs_oper 17543    1 03:51:21 /IN/service_packages/SLEE/bin/mapGenIF
1 SLEE acs_oper 17546    1 03:51:21 N/service_packages/SLEE/bin/xmlTcapInterface
1 SLEE acs_oper 17548    1 03:51:21 /IN/service_packages/SLEE/bin//watchdog
0 SLEE acs_oper process not found: vssp
total processes found = 27 [ 28 expected, 1 not found ]
================================= run-level 3 ==================================
```

Note: The vssp process has not be found in this example indicating that it is not running. A further manual check for the vssp process could be made using the UNIX ps command to verify the output.

```
$ps -ef | grep vssp
acs_oper 19976  1586   0 03:59:29 pts/2       0:00 grep vssp
$
```

Example monitor SLEE resources

To monitor the internal SLEE resource usage with **check**, enter:

[acs_oper] **slee-ctrl> check 1 5**

```
SLEE: Using shared memory offset: 0xc0000000
04:16:29        Dialogs Apps    AppIns  Servs   Events  Calls
                [70000] [30]    [251]   [30]    [207062]        [25000]
04:16:29        70000   24      240     16      207051  25000
04:16:30        70000   24      240     16      207051  25000
04:16:31        70000   24      240     16      207051  25000
04:16:32        70000   24      240     16      207051  25000
04:16:33        70000   24      240     16      207051  25000
 [acs_oper] slee-ctrl>
```

Note: The check command is a separate SLEE utility located in **/IN/service_packages/SLEE/bin** directory. Use `check -h` to see a brief description of usage. For more detail on the check utility refer to the *SLEE Technical Guide*.

Corrupt memory symptom

If the SLEE's shared memory becomes corrupted, for whatever reason, you may find, when you try stopping the SLEE, it will hang.

As a rule of thumb, if it takes longer that 15 seconds to stop the SLEE, it is quite likely to have hung. You need to press **Ctrl C** to return to the command line prompt.

If you come across this scenario, use the `slee-ctrl stop abort` command. This will send a terminate signal (kill -TERM) to any running SLEE processes, wait 3 seconds, then send a kill signal (kill -KILL) to any remaining SLEE processes still running (if any). For example:

```
[ebe_oper] slee-ctrl> stop abort
```

Always try the stop command first before trying the stop abort command.

Run levels

The servers that have the SLEE component installed are configured with a **/etc/rc3.d** start script and **/etc/rc1.d** stop script.

This means that on boot up the server should come up and automatically move to a state where it can handle traffic without operator intervention. You may notice that unlike the init managed processes the SLEE will not be stopped in run-level 2. This is done to prevent any unnecessary stopping of the SLEE that could cause interruption to live traffic.

Warning: If reconfiguring these rc scripts then you *must* follow this rule:

The SLEE must be started after the database and must be stopped before the database.

If the SLEE is not stopped before the database then the database shutdown will hang due to SLEE processes having open shadow connections.

Database Management

Introduction

The database is an integral part of the NCC solution and the majority of the NCC applications have a dependency on it being available before they are able to start.

The solution design has the SMS database as the master data store. The SLC and VWS databases are replicated nodes containing the same data, or in some cases subset of data, that the master SMS node contains.

Currently this is a NCC specific application and is often simply referred to as replication.

Note: The VWS database also has its own set of special tables (BE_% tables) that are not part of the replicated set of SMS tables. The VWS BE_% tables contain a near real-time persistent store of billing data.

Oracle System IDs

This table shows the unique instance names or Oracle System IDs (ORACLE_SID) of each server database.

Server	Oracle SID
SMS	SMF
SLC	SCP
VWS	E2BE

Database verification

The simple way to verify the database is okay and that the NCC processes can connect to it is to start a sqlplus session, enter:

```
$ su – smf_oper
Password:
$ sqlplus /

SQL*Plus: Release 12.1.0.2.0 - Production on Wed Apr 26 21:00:41 2016

Copyright (c) 1982, 2014, Oracle.  All Rights Reserved.


Connected to:
Oracle Database 12c Release 12.1.0.2.0 - 64bit Production

SQL> quit
Disconnected from Oracle Database 12c Release 12.1.0.2.0 - 64bit Production
$
```

The database on each server component is configured to start at boot time by the init daemon when the Server passes through run-level 2 using the **/etc/rc2.d/S99oracle** script.

Note: As discussed earlier, the NCC processes are configured to start in run-level 3. This ensures that the database is available before the NCC processes start. It also has the added advantage that going to run-level 2 stops the NCC processes but does not shut down the database.

Shadow connections

If the database needs to be shut down for maintenance, before this can happen, *all connections to the database* (often referred to as shadow connections) must be disconnected first.

This means that all the NCC processes (init and SLEE managed) must be shut down.

An easy way to check for shadow connections to the database (without connecting to the database) is using the UNIX `ps` command. Enter:

```
$ ps -ef | grep oracle.*LOCAL
  oracle  1116     1   0   Oct 04 ?           0:00 oracleSCP (LOCAL=NO)
  oracle  1106     1   0   Oct 04 ?           0:29 oracleSCP (LOCAL=NO)
  oracle 19564     1   0   Oct 25 ?           0:00 oracleSCP (LOCAL=NO)
  oracle 17552     1   0 03:51:21 ?           0:00 oracleSCP (LOCAL=NO)
  oracle  1110     1   0   Oct 04 ?           0:00 oracleSCP (LOCAL=NO)
...skipped...
$
```

Note: The Parent PID (PPID) are always 1 for shadow connections and in this example the process name, oracleSCP, shows the SID of the database indicating it is a SLC server.

Database startup

Follow these steps to start the database (and listeners) and NCC components. This is simply a reverse of the database shutdown procedure.

Step	Action
1	Start the database by using one of the following methods:

Method 1

```
$ su -
Password:
$ /etc/init.d/oracle start
```

Starting Oracle:

```
Sourcing /etc/profile.ORA

LSNRCTL for Linux: Version 19.0.0.0.0 - Production on 05-MAR-2025 14:23:05

Copyright (c) 1982, 2014, Oracle.  All rights reserved.

Starting /u01/app/oracle/product/11.2/bin/tnslsnr: please wait...

TNSLSNR for Linux: Version 19.0.0.0.0 - Production
System parameter file is /u01/app/oracle/product/11.3/network/admin/listener.ora
Listening on: (DESCRIPTION=(ADDRESS=(PROTOCOL=tcp)(HOST=testslc)(PORT=1521)))

Connecting to (DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(PORT=1521)))
STATUS of the LISTENER
------------------------
Alias                     LISTENER
Version                   TNSLSNR for Linux: Version 19.0.0.0.0 - Production
Start Date                27-OCT-2010 23:13:59
Uptime                    0 days 0 hr. 0 min. 0 sec
Trace Level               off
Security                  ON: Local OS Authentication
SNMP                      OFF
Listener Parameter File   /u01/app/oracle/product/11.3/network/admin/listener.ora
Listening Endpoints Summary...
  (DESCRIPTION=(ADDRESS=(PROTOCOL=tcp)(HOST=testslc)(PORT=1521)))
Services Summary...
Service "SCP" has 1 instance(s).
  Instance "SCP", status UNKNOWN, has 1 handler(s) for this service...
The command completed successfully
Sourcing /etc/profile.ORA
Processing Database instance "SCP": log file
/u01/app/oracle/product/11.3/startup.log
#
* Note: another valid db startup method is
```

Method 2

Change to the oracle user and run the dbstart command, as follows:

```
$ su – oracle
$ dbstart $ORACLE_HOME
```

as above...

Step	Action

2	You would expect to see the following running database processes:
	`$ ps -ef | grep ora_`
	`oracle 33 1 0 23:21:18 ? 0:01 ora_pmon_SCP`
	`oracle 43 1 0 23:21:18 ? 0:01 ora_ckpt_SCP`
	`oracle 49 1 0 23:21:18 ? 0:01 ora_mmon_SCP`
	`root 117 29059 0 23:43:47 pts/2 0:00 grep ora_`
	`oracle 58 1 0 23:21:31 ? 0:02 ora_cjq0_SCP`
	`oracle 35 1 0 23:21:18 ? 0:00 ora_psp0_SCP`
	`oracle 51 1 0 23:21:18 ? 0:00 ora_mmnl_SCP`
	`oracle 47 1 0 23:21:18 ? 0:00 ora_reco_SCP`
	`oracle 39 1 0 23:21:18 ? 0:00 ora_dbw0_SCP`
	`oracle 45 1 0 23:21:18 ? 0:01 ora_smon_SCP`
	`oracle 37 1 0 23:21:18 ? 0:01 ora_mman_SCP`
	`oracle 41 1 0 23:21:18 ? 0:00 ora_lgwr_SCP`
	`#`

3	You may verify the database instance is accepting connections:
	`$ su - smf_oper -c "echo exit |sqlplus /"`
	`Sourcing /etc/profile.ORA`
	
	`SQL*Plus: Release 12.1.0.2 - Production on Wed Apr 26 23:48:21 2016`
	
	`Copyright (c) 1982, 2014, Oracle Corporation. All rights reserved.`
	
	`Connected to:`
	`Oracle Database 11g Release 12.1.0.2 - 64bit Production`
	
	`SQL> Disconnected from Oracle Database 11g Release 12.1.0.2 - 64bit Production`
	`#`

4	Starting the remaining NCC components should occur (dependent on SLEE **rc?.d** script rules) by putting the server back into run level 3. `$ init 3`

Shutting down the Server

The procedure to shut down a server is not be discussed in great detail here as System Administrators generally have their own procedures that they like to follow to do this.

What is suggested is a method that may be followed.

It is recommended that the system administrators who maintain the NCC servers create and test a detailed server shutdown procedure that meets their needs.

The general steps include:

Step	Action
1	Shut down the NCC applications and database (as previously described).
2	Make a connection to the server console (specific to the Server and network setup).
3	Issue a server shutdown command to stop the operating system (for example init 0).
4	Power server off (refer to Server documentation).

Starting the Server

The procedure to start a server can be dependent on the setup of the server and the procedures that system administrators may like to use for the servers they maintain.

Standard practice for the NCC servers would be for the operating system to boot up when the server is powered on.

Chapter 3

Monitoring and Managing

Overview

Introduction

This chapter gives some hints and tips for system administrators to manage the NCC product.

In this chapter

This chapter contains the following topics.

Monitoring and Managing Overview

Overview

For system administrators new to the NCC solution, the monitoring and managing of the solution can seem a little daunting due to the large number of different processes and potentially complex interactions with different network types and services.

Like anything though, the more you use it and the more familiar it becomes and the easier it gets.

The aim of this chapter is to introduce some common tools and aids to help administrators become more comfortable in monitoring the solution without getting involved with the technical detail of each component and its operation.

Software Version Levels

Base components

The NCC base components, or applications, and NCC application patches are installed on the system using the operating system commands.

Running Processes

Overview

In the UNIX world a running application, or program, is referred to as a process.

Basically a process is the active instance of the running program that the operating system is allocating system resources to, allowing it to execute instructions.

In the UNIX environment the ps command (see `man ps`) is used to generate a snapshot report of the current status of process(es) and hence how you can tell that an application is running.

Complex environments

In a complex environment, where multiple running processes making up an application suite, the trick is knowing which processes to identify.

To do this you would need to distinguish each NCC daemon that is configured to run in the SLEE configuration file (**SLEE.cfg**) and the **/etc/inittab** file.

The use of start-up scripts can further complicate this task, requiring each script to be checked for the executable binary file name.

To simplify this task the NCC support tools package provides a utility called pslist that, among other things, will identify and quickly verify the status of the NCC processes running on the platform.

pslist command

The pslist command uses a process list configuration (.plc) file containing regular expressions of the relevant NCC processes that are matched against the ps -ef command output. Enter:

```
$ pslist
```

Response:

```
----------------------- Tue Nov 23 01:03:46 GMT 2010 -------------------------
C APP  USER       PID PPID  STIME COMMAND
1 ACS  acs_oper  1004    1 04-Oct /IN/service_packages/ACS/bin/acsCompilerDaemon
1 ACS  acs_oper  1008    1 04-Oct IN/service_packages/ACS/bin/acsProfileCompiler
1 ACS  acs_oper  7553    1 28-Oct rvice_packages/ACS/bin/acsStatisticsDBInserter
1 OSD  acs_oper  1047    1 04-Oct IN/service_packages/OSD/bin/osdWsdlRegenerator
1 CCS  ccs_oper  1011    1 04-Oct /IN/service_packages/CCS/bin/ccsCDRLoader
1 CCS  ccs_oper  1033    1 04-Oct N/service_packages/CCS/bin/ccsCDRFileGenerator
2 CCS  ccs_oper  1406 1043 04-Oct /IN/service_packages/CCS/bin/ccsProfileDaemon
1 CCS  ccs_oper 20252    1 28-Oct /IN/service_packages/CCS/bin/ccsBeOrb
1 CCS  ccs_oper  9413    1 04-Oct /IN/service_packages/CCS/bin/ccsChangeDaemon
1 EFM  smf_oper   995    1 04-Oct /IN/service_packages/EFM/bin/smsAlarmManager
1 PI   smf_oper  1080    1 04-Oct /IN/service_packages/PI/bin/PImanager
6 PI   smf_oper  1319 1080 04-Oct PIprocess
1 PI   smf_oper  9186 1080 04-Oct PIbeClient
2 SMS  smf_oper  6173    1 26-Oct /IN/service_packages/SMS/bin/smsMaster
1 SMS  smf_oper   943    1 04-Oct /IN/service_packages/SMS/bin/smsNamingServer
1 SMS  smf_oper   944    1 04-Oct /IN/service_packages/SMS/bin/smsReportsDaemon
1 SMS  smf_oper   946    1 04-Oct IN/service_packages/SMS/bin/smsReportScheduler
1 SMS  smf_oper   947    1 04-Oct /IN/service_packages/SMS/bin/smsAlarmDaemon
1 SMS  smf_oper   948    1 04-Oct /IN/service_packages/SMS/bin/smsStatsThreshold
1 SMS  smf_oper   949    1 04-Oct /IN/service_packages/SMS/bin/smsTaskAgent
1 SMS  smf_oper   969    1 04-Oct /IN/service_packages/SMS/bin/smsTrigDaemon
2 SMS  smf_oper   979    1 04-Oct /IN/service_packages/SMS/bin/smsConfigDaemon
1 SMS  smf_oper   980    1 04-Oct /IN/service_packages/SMS/bin/smsStatsDaemonRep
total processes found = 31 [ 31 expected ]
================================= run-level 3 ==================================
```

Note: The listed output is from a SMS platform and is grouped by user and application components.

Default plc file

When a UNIX user first runs the pslist command without command line options it will automatically generate a default **.plc** file by scanning the **/etc/inittab** and **/IN/service_packages/SLEE/etc/SLEE.cfg** file, if existing.

The user's default .plc file can be viewed with the pslist -v command line option. Enter:

```
$ pslist –v
```

Response:

```
[ /IN/service_packages/ACS/tmp/ps_processes.telco-p-slc01.plc ]
#########################################################################
# pslist: default process list configuration (plc) file used to match and  #
# display running processes.                                              #
# File creation time: Tue Nov 23 01:58:39 GMT 2010                        #
# Lines beginning with a hash (#) character are ignored.                  #
# $1="grouped-apps name (max 5-char)" $2="regex of process" [$3+=comments] #
#########################################################################
ACS   acs_oper.*\/IN\/service_packages\/ACS\/bin\/acsStatsMaster               inittab
ACS   acs_oper.*\/IN\/service_packages\/ACS\/bin\/cmnPushFiles                 inittab
SMS   smf_oper.*\/IN\/service_packages\/SMS\/bin\/cmnPushFiles                 inittab
SMS   smf_oper.*\/IN\/service_packages\/SMS\/bin\/smsAlarmDaemon               inittab
SMS   smf_oper.*\/IN\/service_packages\/SMS\/bin\/smsConfigDaemon$             inittab
SMS   smf_oper.*\/IN\/service_packages\/SMS\/bin\/smsStatsDaemon               inittab
SMS   smf_oper.*\/IN\/service_packages\/SMS\/bin\/updateLoader                 inittab
SMS   smf_oper.*bin\/(smsC|c)+ompareResync(Client|Recv)+     inittab: Update Loader child
process
UIS   uis_oper.*\/IN\/service_packages\/UIS\/bin\/UssdMfileD                   inittab
UIS   uis_oper.*\/IN\/service_packages\/UIS\/bin\/cmnPushFiles                 inittab
XMS   smf_oper.*\/IN\/service_packages\/XMS\/bin\/oraPStoreCleaner             inittab
SLEE  .*_oper.*\/IN\/service_packages\/ACS\/bin\/acsStatsLocalSLEE
SLEE  .*_oper.*\/IN\/service_packages\/ACS\/bin\/slee_acs
SLEE  .*_oper.*\/IN\/service_packages\/E2BE\/bin\/BeClient
SLEE  .*_oper.*\/IN\/service_packages\/SLEE\/bin\/\/watchdog
SLEE  .*_oper.*\/IN\/service_packages\/SLEE\/bin\/alarmIF
SLEE  .*_oper.*\/IN\/service_packages\/SLEE\/bin\/replicationIF
SLEE  .*_oper.*\/IN\/service_packages\/SLEE\/bin\/timerIF
SLEE  .*_oper.*\/IN\/service_packages\/SLEE\/bin\/xmlTcapInterface
ps_processes.telco-p-slc01.plc: END
 [Press space to continue, q to quit, h for help]
```

SLEE against Service Daemons

There is an important distinction to make for the processes that are managed by either the service daemon or SLEE.

Turn on and off the process **/etc/inittab** file by running the following command:

```
init q
```

When started, the SLEE creates a common backbone that ties a group of applications together

Note: You can individually configure the processes to run except for SLEE managed processes.

pslist SLEE only example

The following pslist example is taken on a VWS and shows running SLEE processes, but no running init managed processes. Enter:

```
$ pslist
```

Response:

```
----------------------- Tue Nov 23 02:48:46 GMT 2010 -------------------------
 C APP  USER       PID PPID    STIME COMMAND
 1 SLEE ebe_oper 16139    1 02:23:25 /IN/service_packages/SLEE/bin/timerIF
```

```
16 SLEE ebe_oper 16143    1 02:23:25 /IN/service_packages/E2BE/bin/beVWARS
 1 SLEE ebe_oper 16156    1 02:23:25 /IN/service_packages/E2BE/bin/beSync
 1 SLEE ebe_oper 16157    1 02:23:25 /IN/service_packages/E2BE/bin/beServer
 4 SLEE ebe_oper 16166    1 02:23:25 /IN/service_packages/E2BE/bin/beGroveller
 1 SLEE ebe_oper 16178    1 02:23:26 /IN/service_packages/SLEE/bin/replicationIF
 1 SLEE ebe_oper 16179    1 02:23:26 /IN/service_packages/DAP/bin/dapIF
 1 SLEE ebe_oper 16182    1 02:23:26 /service_packages/E2BE/bin/beEventStorageIF
 1 SLEE ebe_oper 16183    1 02:23:26 /service_packages/E2BE/bin/beServiceTrigger
 1 SLEE ebe_oper 16184    1 02:23:26 ervice_packages/CCS/bin/ccsSLEEChangeDaemon
 1 SLEE ebe_oper 16185    1 02:23:26 /IN/service_packages/SLEE/bin//watchdog
 0 CCS  Did not match regex: /ccs_oper.*.\/bin\/updateLoader( |$)+/
 0 CCS  Did not match regex: /ccs_oper.*\/IN\/service_packages\/CCS\/bin\/ccsMFileCompiler(
|$)+/
 0 CCS  Did not match regex: /ccs_oper.*bin\/(smsC|c)+ompareResync(Client|Recv)+( |$)+/
 0 CCS  Did not match regex: /ccs_oper.*cmnPushFiles( |$)+/
 0 E2BE Did not match regex: /ebe_oper.*\/IN\/service_packages\/E2BE\/bin\/beCDRMover( |$)+/
 0 E2BE Did not match regex: /ebe_oper.*cmnPushFiles( |$)+/
 0 SMS  Did not match regex: /smf_oper.*\/IN\/service_packages\/SMS\/bin\/smsAlarmDaemon( |$)+/
 0 SMS  Did not match regex: /smf_oper.*\/IN\/service_packages\/SMS\/bin\/smsConfigDaemon$/
 0 SMS  Did not match regex: /smf_oper.*\/IN\/service_packages\/SMS\/bin\/smsStatsDaemon( |$)+/
total processes found = 29 [ 38 expected, 9 not found ]
================================= run-level 2 ==================================
```

Note: The count column (C) is 0 for the processes that were not matched and the total processes found output highlights that the expected number of processes were not matched. In this case there is a clue on the bottom line as to why the init managed processes are not running.

Expected against Not Found processes

It is worthwhile noting that the expected and not found counts are only an estimate of the number of processes missing.

Some processes may spawn multiple child processes that may have the same process name (for example, PI processes on the SMS) making it impossible to accurately predict how many processes are expected or not found.

Recreate default plc file

If changes are made to the platform and the running services then it may be necessary to re-create the default **.plc** file.

You can do this using the pslist -d option.

Warning: Only processes that are configured to run will be added to the default **`.plc`** file. For example, if a **/etc/inittab** process is commented out, or set to off, then when the pslist-d command is run the commented entry will not be added to the list.

Another option is to become root user and run pslist -xy to delete all user's default .plc file. The next time a user runs the pslist command, it will automatically re-create a default **`.plc`** file.

pslist syntax

To see a full list of supported command line options, with a brief description, use `pslist -h` (help) option or `man pslist` for full documentation on pslist usage.

Syntax:

```
pslist -[acCehiqrRstvxyz] [-d (esg|slee|init)] [-S SLEE config] [-f .plc file] [-u
user] [-k (1|9|15)] [-l (1|2)] [regex (app-name|process name)]
```

pslist parameters

This table gives a brief description of the supported options.

Parameter	Description
-d	Creates the default process list configuration [**~***user*/**tmp/ps_processes.***hostname***.plc**] file. Valid *hostname* arguments are: • esg scan both the inittab and SLEE config files (default) • slee scan the defined SLEE configuration file • init[tab] scan the /etc/inittab file
-r	Displays system resource use information (for example. %cpu, %memory).
-s	Scans the SLEE config file and prints the status of the matched process(es).
-i	Scans **/etc/inittab** file for start-up scripts and prints the status of the matched process(es).
-e	Scans both the defined SLEE config file and the **/etc/inittab** file and prints the status of the matched process(es).
-c	Clustered SMS option that creates a temporary .plc file to list cluster managed processes (requires scstat command).
-f	Specify a process list configuration (.plc) file different to the default one. [**~***user*/**tmp/ps_processes.***hostname***.plc**].
-S	Specify a different SLEE config file.
-u	Specify a different user as the process owner (can be regex).
-C	The output in the COMMAND column is not trimmed so the full command line is displayed.
-a	Displays the command line arguments of matched process(es) (SunOS only).
-q	No output in quiet mode. The exit value is the count of the matched processes (up to a maximum of 99).
-t	Displays a process tree of PPID and COMMAND (SunOS only).
-v	View the process list configuration (.plc) file.
-x	Ask to remove any default process list configuration files (**ps_processes.***hostname***.plc) found in the /tmp, ~***user*/tmp**, and **~<*_oper>/tmp** directories.
-k	Ask if one of the following kill signals should be sent to all matched processes. Valid arguments are: 1 SIGHUP 9 SIGKILL 15 SIGTERM
-y	Specify yes. Use with -x to confirm remove, or -k option to confirm kill.
-l	Adds an entry to the system log with the expected and actual process count info.
-z	Creates a softlink for the shorter "pslist" command in directory **/usr/local/bin**. **Note:** Users must have that directory set in their PATH environment variable for it to work.
-R	Displays pslist revision info.
-h	Displays the above usage text.

Creating own plc file

As can be seen, there are multiple command line options and the more experienced administrator may like to make their own **.plc** file (-f option) with an alias command to list other important processes.

For example, you could create a command to list the Oracle database processes on the VWS.

```
$ cat <<EOF >ora.plc
 ORA ora_.*E2BE      core db processes
 ORA oracleE2BE      shadow process
 EOF
$ alias psdb='pslist -f ora.plc $*' # note: add this line to your user
~/.profile.
$ psdb -r
----------------------- Tue Nov 23 21:28:57 GMT 2010 -------------------------
APP USER     PID PPID S %CPU %MEM     VSZ     RSS  TIME     ELAPSED COMMAND
ORA oracle   744    1 S  0.0 51.3 4319336 4091784 30:40 50-07:34:48 ora_pmon_E2BE
ORA oracle   746    1 S  0.0 51.2 4317864 4090304 02:13 50-07:34:48 ora_psp0_E2BE
ORA oracle   748    1 S  0.0 51.2 4317864 4090528 02:41 50-07:34:48 ora_mman_E2BE
ORA oracle   750    1 S  0.0 51.3 4323928 4095976 05:02 50-07:34:48 ora_dbw0_E2BE
ORA oracle   752    1 S  0.0 51.2 4330840 4091264 03:47 50-07:34:48 ora_lgwr_E2BE
ORA oracle   754    1 S  0.0 51.2 4319928 4091472 45:09 50-07:34:48 ora_ckpt_E2BE
ORA oracle   756    1 S  0.0 51.3 4318952 4092184 04:40 50-07:34:48 ora_smon_E2BE
ORA oracle   758    1 S  0.0 51.2 4317928 4090904 00:03 50-07:34:48 ora_reco_E2BE
ORA oracle   760    1 S  0.0 51.3 4319720 4092048 04:59 50-07:34:48 ora_mmon_E2BE
ORA oracle   762    1 S  0.0 51.2 4317928 4090744 02:32 50-07:34:48 ora_mmnl_E2BE
ORA oracle 16159    1 S  0.0 46.1 4317920 3681768 00:00    19:05:33 oracleE2BE
ORA oracle 16161    1 S  0.0 46.1 4317920 3681768 00:00    19:05:33 oracleE2BE
ORA oracle 16163    1 S  0.0 46.1 4317920 3681768 00:00    19:05:32 oracleE2BE
ORA oracle 16168    1 S  0.0 46.1 4317920 3681768 00:00    19:05:32 oracleE2BE
ORA oracle 16170    1 S  0.0 46.1 4317920 3681768 00:00    19:05:32 oracleE2BE
ORA oracle 16172    1 S  0.0 46.1 4317920 3681768 00:00    19:05:32 oracleE2BE
ORA oracle 16175    1 S  0.0 46.1 4317920 3681768 00:00    19:05:32 oracleE2BE
ORA oracle 16177    1 S  0.0 46.1 4317920 3681768 00:00    19:05:32 oracleE2BE
ORA oracle 16181    1 S  0.0 46.1 4317920 3681768 00:00    19:05:32 oracleE2BE
ORA oracle 16189    1 S  0.0 46.1 4317920 3681768 00:00    19:05:32 oracleE2BE
ORA oracle 16191    1 S  0.0 46.1 4317920 3681768 00:00    19:05:32 oracleE2BE
ORA oracle 16193    1 S  0.0 46.1 4317920 3681768 00:00    19:05:32 oracleE2BE
ORA oracle 16195    1 S  0.0 46.1 4317920 3681768 00:00    19:05:32 oracleE2BE
ORA oracle 16197    1 S  0.0 46.1 4317920 3681768 00:00    19:05:32 oracleE2BE
ORA oracle 16199    1 S  0.0 46.1 4317920 3681704 00:00    19:05:31 oracleE2BE
ORA oracle 16201    1 S  0.0 46.1 4317920 3681704 00:00    19:05:31 oracleE2BE
ORA oracle 16203    1 S  0.0 46.1 4317984 3681832 00:00    19:05:31 oracleE2BE
ORA oracle 16205    1 S  0.0 46.1 4317920 3681768 00:00    19:05:31 oracleE2BE
ORA oracle 16207    1 S  0.0 46.1 4317920 3681768 00:00    19:05:31 oracleE2BE
ORA oracle 16209    1 S  0.0 46.1 4317920 3681704 00:00    19:05:31 oracleE2BE
ORA oracle 16212    1 S  0.0 46.1 4317920 3681704 00:00    19:05:31 oracleE2BE
ORA oracle 16214    1 S  0.0 46.1 4317920 3681704 00:00    19:05:31 oracleE2BE
ORA oracle 16216    1 S  0.0 46.1 4317920 3681768 00:00    19:05:31 oracleE2BE
ORA oracle 16218    1 S  0.0 46.1 4317920 3681704 00:01    19:05:31 oracleE2BE
ORA oracle 16222    1 S  0.0 46.1 4317920 3681832 00:00    19:05:31 oracleE2BE
total processes found = 35
================================ run-level 2 ================================
```

General comment

Generally speaking, pslist is the only command you need to remember when checking for the NCC processes configured to run on any platform and it is a good place to start when investigating or troubleshooting issues.

Note: The slee-ctrl status and resource commands respectively call the pslist -s and pslist -sr commands to generate output.

SLEE Resource Usage

Introduction

SLEE resources are created at run time and are used during service control for passing information between SLEE interfaces and applications.

These are internal values that have no specific meaning to the rest of the network and are entirely local to the running SLEE.

The NCC SLEE provides an application called `check` that can monitor and report on current SLEE resource usage. See *SLEE Technical Guide* for more information on this utility.

SLEE resources

There are three main types of internal SLEE resources;

- Dialogs - exist for the duration of the call.
- Events - these are transient and are deleted upon delivery to the relevant SLEE entity.
- Calls

If you have a networking background then you may notice a similarity to to the SS7 model here. In fact, at a high level, the SLC SLEE considers everything that hits the platform as a Call instance, be it a voice or data call, SMS, or some sort of business process logic.

Like the SS7 model, a call instance has a source and destination address for a communication channel or dialog to pass data or events back and forth.

For example, a new service request hits a SLEE interface generating a new call instance.

The SLEE Interface will create a dialog to the relevant SLEE application and the call context data is put into an event and sent (through the dialog) to the SLEE application for processing.

The SLEE application can create new dialogs to other SLEE interfaces or applications for the call instance.

Resource snapshot

The following is an example of typical SLEE resource usage snapshots taken on a SLC at one second intervals.

```
/IN/service_packages/SLEE/bin$ check 1 5
22:24:02    Dialogs  Apps    AppIns        Servs        Events   Calls
            [70000]  [10]     [101]         [10]       [190242] [25000]
22:24:02    61753     9        90            0          196170    21874
22:24:03    61797     9        90            0          196164    21892
22:24:04    61845     9        90            0          196164    21900
22:24:05    61841     9        90            0          196170    21900
22:24:06    61886     9        90            0          196192    21909
```

Notes:

- The dialog, events, and calls values changing.
- The output values, in square brackets, on the second row, indicates the maximum SLEE resource values allocated on SLEE start-up.
- You should note that the events value appears to be lower than the first output value. This is a known issue due to the fact Events can be configured to pre-allocated sizes with different values (**SLEE.cfg**) causing the check output value to be miscalculated. This issue can safely be ignored.

SLEE health

The SLC SLEE resource usage in a healthy, well-functioning, network will fluctuate up and down during the course of a day as the call rates and call mix changes.

Note: The SLEE SLC config includes a concept of the max call rate. This is the maximum call activity that can be supported and is used as an input in SLEE.cfg to size resource requirements.

During high or peak call volume, more SLEE resources are used (that is, lower SLEE resource values), but will return back towards their maximum values at low call volume times, such as early morning.

The configured amount of SLEE resource is a lot higher than necessary for normal operation to allow the SLC to cope for a period of time with abnormal network behavior and/or software bugs which can prevent the SLEE resources from being freed up correctly (this situation is often referred to as a call leak).

Normal check output

So what is normal for the SLC check output?

There is no simple answer as every network is different. What is normal for your network can only be found by monitoring the SLC SLEE resource usage over a period of time, taking regular SLEE resource snapshots to create a normal historical trend.

Once a normal pattern of SLEE resource usage has been established, over weeks and months, then anything outside of this normal trend may indicate an issue with either; the NCC software, or, more often than not, the connecting SS7 network itself.

Experience has shown that this usually occurs due to routing failure and/or congestion within the SS7 network elements. The NCC term we use for leaked voice calls is an AWOL (absent without leave) call.

AWOL calls

An AWOL voice call usually occurs when the SLC does not receive an applyChargingReport (ACR) from the SS7 network and therefore does not know whether the call has ended or not. This will tie up the SLEE resource because, as far as the SLC is concerned, the call never ends.

The CAP (CAMEL Application Part) signalling standard does not cover the scenario of stale calls on the SLC where the SLC does not receive a response from the SS7 network for an in progress call.

It can be argued that when a response from the Network does not arrive, after a certain amount of time since the last response, that the SLC could assume it will never receive any more responses and should therefore abort the call, thereby freeing up the SLEE resource.

Another consideration is that most billing reservations have a defined length of validity and call flow transactions within the SS7 network need to be within this time limit also.

As a defense mechanism to this type of stale call scenario, the NCC solution has an optional AWOL setting that allows the SLC to clear out stale calls once they pass a configurable period of time. This allows the finite SLEE resources on the SLC to be freed up, if abnormal SS7 network behavior occurs.

Note: The timeout parameters in the MAP (Mobile Application Part) signaling protocol, used for Short Message Service (SMS), allows the SLC to clear out stale transactions.

Scarce SLEE resources

As a rule of thumb, the following values should indicate a problem condition that may require urgent investigation:

- Serious - should be investigated:
  SLEE dialogs or calls are less than 25% of their maximum value.
- Critical - must be urgently investigated:

SLEE dialogs or calls are less than 10% of their maximum value.

SLEE events are around 50% of their maximum value.

Note: New calls can still be made if there are available dialog and calls resources that are greater than 0. The only time new calls cannot be made is when the free dialogs and/or calls are completely exhausted. At this stage the best cause of action is a quick SLEE restart, to reset the SLEE resources, allowing new calls to be made. If the situation reoccurs after a SLEE restart, review the setting of max call rate if significant additional traffic has been cut over.

Warning messages

The NCC SLEE will generate one warning event, in the **/var/adm/messages** log file, for each SLEE resource that breaks a threshold of 80% of its maximum value, similar to this example:

```
Sep 12 01:36:37 telco-p-slc01 watchdog: [ID 953149 user.warning] watchdog(18965)
WARNING: 20006 Call Instances Locked breaches 80% of total available instances
(25000)
```

If SLEE resources should become exhausted then the **/var/adm/messages** log file will start spewing out entries, similar to the following notice message, for each failed call attempt:

```
Sep 17 03:22:59 telco-p-slc01 xmsTrigger: [ID 167701 user.notice] xmsTrigger(18957)
NOTICE SleeInterfaceAPI=8006 sleeInterfaceAPI.cc@248: Overload: SLEE is out of call
instances
```

Resource leak

If a SLEE resource leak is caught early then it will be the rate of the call leak that determines the true severity of the issue.

A slow leak of days, weeks, or months may be acceptable with general platform maintenance clearing leak calls during SLEE restarts.

A rapid SLEE resource loss over hours, minutes or seconds (depending on current call rates) requires instant action and investigation.

Good call tracing in the signaling network is vital in tracking down where in the network the issue is originating.

Monitoring SLEE resources

The NCC Support Tools package provides the **check-SLEE.sh** script that, among other things, can be used for generating a timestamped log file of SLEE resource usage.

It is designed to run from the SLC acs_oper user's crontab file. If the following entry does not exist, add it with the crontab command, as below:

```
acs_oper@telco-p-slc01$ crontab -e
...skipped...
# the following entry is logging internal SLEE resource usage
* * * * * ( . /IN/service_packages/ACS/.profile ;
/IN/service_packages/SUPPORT/bin/check-SLEE.sh ) >>
/IN/service_packages/SUPPORT/tmp/check-SLEE.log 2>&1
```

Tip: One minute snapshot intervals are recommended (as highlighted).

check-SLEE.sh output

This example output is generally what you will see on a well-functioning healthy network with SLEE resources adjusting up and down by small margins over the course of a minute.

The output of the **check-SLEE.sh** script is written to the **check-Time_Dialogs_Events_Calls_CAPS.log** file in the **/IN/service_packages/SUPPORT/tmp** directory. You can view this log with ckslee command as follows:

```
$ ckslee
       Date-Time Diags Events Calls[Change]      CAPS
20101125-03:43:00 86889 253630 43287[+0.04%]        5
20101125-03:44:01 86904 253630 43300[+0.05%]        5
20101125-03:45:00 86905 253631 43283[-0.07%]        5
20101125-03:46:00 86911 253627 43297[+0.06%]        6
20101125-03:47:00 86924 253632 43296[-0.00%]        5
20101125-03:48:00 86870 253629 43264[-0.13%]        5
20101125-03:49:00 86926 253631 43296[+0.13%]        5
20101125-03:50:00 86945 253630 43306[+0.04%]        4
20101125-03:51:00 86908 253627 43293[-0.05%]        4
20101125-03:52:00 86938 253628 43302[+0.04%]        5
```

Notes:

The [Change] value is a measure percentage difference between the previous Calls value and is useful in highlighting a sudden change resource usage.

The CAPS (Call attempts per second) column is an extra, but requires the SIGTRAN stack(s) to be configured to output their average CAPS rate. If they are not configured, this column will show `N/A`. See NCC *SIGTRAN Technical Guide* and the `monitorperiod`, `rejectlevel`, `reportperiod`, and `displaymonitors` parameters to enable CAPS reporting.

check-SLEE.sh archiving

Problems with SLEE resources will usually be found when the solution is first installed, or when the underlying signaling network has changes made to it.

This is when having a historical log of SLEE resource usage over time is a great tool in determining the trend, or a trigger point to an issue occurring.

To archive the **check-Time_Dialogs_Events_Calls_CAPS.log** file, add the following line to the **/IN/service_packages/ACS/etc/logjob.conf** file.

```
log /IN/service_packages/SUPPORT/tmp/check-Time_Dialogs_Events_Calls_CAPS.log age
240 size 1M arcdir /IN/service_packages/SUPPORT/tmp/archive
```

check-SLEE.sh usage

The head of the check-SLEE.sh script is extensively commented and explains some other ways the check-SLEE.sh script can be used, such as automatic SLEErestarts if SLEE resources go below a defined threshold.

VWS SLEE resources

As all the VWS processes are SLEE Interfaces, the SLEEresource usage pattern is different to the SLC.

On SLEE start-up, dialogs are created between the different SLEE interfaces for events to be passed back and forth. However the operations on the VWS are purely transactional and the concept of a call instance is not necessary. Therefore only the events value ever changes, with the dialogs and calls values remaining static as shown here:

```
$ slee-ctrl check 1 5

SLEE Control: v,1.0.10: script v,1.19: functions v,1.46: pslist v,1.118

[ebe_oper] slee-ctrl> check 1 5
SLEE: Using shared memory offset: 0xc0000000
03:05:06        Dialogs  Apps    AppIns     Servs      Events   Calls
               [1000]    [10]     [100]      [10]      [71250]  [500]
03:05:06         964       10       100        10        71252   500
03:05:07         964       10       100        10        71253   500
03:05:08         964       10       100        10        71255   500
```

```
03:05:09        964       10       100        10        71254   500
03:05:10        964       10       100        10        71255   500
```

Note: You can use the slee-ctrl command to call the check program.

Unless advised to, the monitoring of VWS SLEE resource usage is of little benefit.

Rolling Snoop Archives

Introduction

The NCC solution sits between the SS7 signaling network and LAN with Network Connectivity Agents (NCA) providing interfaces into the SLEE to interpret, convert and retransmit various network protocols, such as SIGTRAN (SS7 over IP), DIAMETER, SOAP/XML, SMPP and internal messaging protocols.

Being able to capture the reality of what is actually being sent over the wire is a vital tool when analyzing NCA related issues to help identify the source of the problem, be it local or remote.

Running the tshark/snoop utility allows you to trace and capture incoming and outgoing traffic passing through a server's Network Interface Card (NIC) Devices.

To aid in the tracing of network devices and the retention of their snoop capture files the Support Tools package provides, what has become known as, the rolling snoop scripts.

Scripts

You can find the latest version of the rolling snoop scripts in the latest Support Tools package which are installed in the **/IN/service_packages/SUPPORT/bin** directory.

The **rolling-snoop.sh** and **start-rolling-snoop.sh** scripts are heavily commented and it is advisable to read these to get a better understanding of the scripts and their configuration.

For ease of use and convenience it is recommended that the **start-rolling-snoop.sh** and **stop-rolling-snoop.sh** wrapper scripts are used to control the rolling-snoop.sh script even though it may be run independently.

rolling-snoop.sh

The script that runs the snoop command.

Responsible for checking disk space, and archiving capture files before exiting.

The command line arguments allow you to specify a capture file name prefix and the ability to pass in valid snoop command line expressions to filter packets on, such as IP traffic that is either; sctp or tcp protocol, on port x or host y (use `$ man snoop` for more details of valid expressions).

Usage:

`rolling-snoop.sh [capture file prefix:]`*`network_card_interface_device_name`*
`[snoop options]`

For example:

```
rolling-snoop.sh e1000g2
rolling-snoop.sh uas01-sig-pri:e1000g2 sctp port 14000
rolling-snoop.sh usms2-chrg-sec:nxge1 -c 1000000 tcp port 3989
rolling-snoop.sh usms01a-mgmt-pi:e1000g0 not icmp port 2999 or port 3000
```

Note: Must be run as root super-user.

start-rolling-snoop.sh

This is a wrapper script where you configure all the network devices you want to snoop, which are then passed as command line parameters to the **rolling-snoop.sh** script.

Each configured NIC will start a **rolling-snoop.sh** script, which in turn starts and controls the the snoop command. The example section below is where you define the network device(s) to trace.

```
/IN/service_packages/SUPPORT/bin$ vi  start-rolling-snoop.sh
...skipped...
cat << CONFIG_END |sed '/^ *#/d; s/^ *//; s/\(.*\)\(#.*\)/\1/; /^ *$/d' > $TEMP_FILE

### CONFIGURE NETWORK DEVICES HERE ###
# configuration examples below:
# e1000g1 # comments ignored after hash
# ${HOST}-sig-pri:e1000g1
# ${HOST}-sig-sec:nxge1 $PACK_COUNT sctp port 14000
# uas01-chrg-pri:e1000g0 -c 250000 tcp host charging-gw port 3989

CONFIG_END
...
```

stop-rolling-snoop.sh

When the **stop-rolling-snoop.sh** is run, all snoop processes are terminated (independently started snoop commands will also be killed), quickly followed by the termination of the **rolling-snoop.sh** scripts.

It is the stopping of the **rolling-snoop.sh** script that manages the archiving of the current capture files. So by regularly stopping and starting the rolling-snoop.sh script, we can easily create an archived repository of snooped traffic.

snoop_archiver.sh

This is a wrapper script to run the **start-rolling-snoop.sh** and **stop-rolling-snoop.sh** scripts and manage the removal of old archived capture files.

This script can be configured, as an hourly root crontab job, thereby creating an archive repository of capture files in hourly timestamped directories.

Here is an example **snoop_archiver.sh** script:

```
/IN/service_packages/SUPPORT/bin$ cat snoop_archiver.sh
#!/bin/ksh
#########################################################################
# Revision: : snoop_archiver.sh,v 1.4 2010/01/05 01:55:10 gcato Exp $
#
# script to regularly archive rolling snoops files
# see rolling_snoop.sh KEEP_SECONDS variable to define archiving frequency
# (usually 60 minutes intervals)
#
# run from root crontab
# 59 * * * * /IN/service_packages/SUPPORT/bin/snoop_archiver.sh >/dev/null 2>&1
#
#########################################################################

# how long to keep archived snoop files before deleting
KEEP_DAYS=3

# snoops archive directory
SNOOP_DIR=/IN/service_packages/SUPPORT/snoops/archive

# snoop directory suffix (usually TZ variable value)
SUFFIX=GMT

# stop all snoops (this will automatically archive the files)
/IN/service_packages/SUPPORT/bin/stop-rolling-snoop.sh

sleep 1
```

```
# restart the rolling snoops
/IN/service_packages/SUPPORT/bin/start-rolling-snoop.sh

# delete old archived snoop dirs
find ${SNOOP_DIR} -type d -name \*${SUFFIX} -mtime +${KEEP_DAYS} |xargs rm -rf

# compress snoop files
find ${SNOOP_DIR}/*${SUFFIX}/ \( -name \*snoop -a ! -name \*gz \) |xargs nice gzip 2>/dev/null
```

Default directory

The default output directory is **/IN/service_packages/SUPPORT/snoops/(current|archive)**. The current directory contains the capture files currently being written to and archive directory contains time-stamped directories with the saved capture files.

Analyzing the Capture Files

A search of the web will provide a list of different protocol analyzer products, such as Wireshark, that can be used to view the capture file data. Please see the documentation of your protocol analyzer of choice for more information on interpreting the output.

Rolling Snoop Risks

Introduction

When running rolling snoop there are potential problems inherent with any data capture tool. This topic covers the major "look out for" issues.

Missing packets

Watch out for the routing of packets through secondary, or fail-over, NIC devices, which are configured in a multipathing group. You will need to snoop both network interfaces to capture all incoming and outgoing traffic.

```
# man ifconfig

...skipped...

 MULTIPATHING GROUPS

     Physical interfaces that share the same IP broadcast domain
     can be collected into a multipathing group using the group
     keyword. Interfaces assigned to the same multipathing group
     are  treated  as  equivalent  and outgoing traffic is spread
     across the interfaces on a per-IP-destination basis.  In
     addition, individual interfaces in a multipathing group are
     monitored for failures; the addresses associated with failed
     interfaces  are automatically transferred to other function-
     ing interfaces within the group.

     For more details on IP multipathing, see in.mpathd(1M) and
    ...<snip>...
```

Basically, just because a packet came in on a network interface does not mean it will go out on the same interface. To find multipathed interfaces use the ifconfig -a command to find the network interfaces that are configured with the same groupname (if any).

Some monitoring and testing will usually show you which interfaces you need to monitor to catch all the traffic you want.

The crontab configured start and stop time will also have a small window of missed packets.

Disk space

The **rolling-snoop.sh** script has a MAX_DISK_PERCENTAGE variable (default 75%) and will not run if the output capture file disk partition exceeds this disk space usage threshold (only checked on start-up and subsequent stop/start of snoop command).

This is to prevent the capture files from taking too much disk space and affecting the Event Data Records and process log files from being created.

Warning Change with extreme caution.

If there is limited disk space then you can either; reduce the KEEP_DAYS variable in the **snoop_archiver.sh** script, or soft-link the **/IN/service_packages/SUPPORT/snoop** directory to a disk with spare capacity. For example:

```
# mkdir /volA/snoops
# rm -r /IN/service_packages/SUPPORT/snoop
# ln -s /volA/snoops /IN/service_packages/SUPPORT/snoops
```

Capture file size

The snoop command does not have a max duration option.

Do not confuse the KEEP_SECONDS variable in the **rolling-snoop.sh** script with how long the snoop command actually runs for.

The MAX_PACKET_COUNT variable (default 100000 packets), also configured in the **rolling-snoop.sh** script, sets the limit to how big a capture file will grow to before a new capture file is started. If there is a lot of traffic on an interface, you may want to decrease this value to keep the capture file to a manageable size.

It is recommended that this is set inside the configurable section of the **start-rolling-snoop.sh** by defining a **-c** option. Further filtering options, on a per device level, can also help keep the capture file size manageable (read the script's comments for more details).

Depending on the amount of IP traffic, you may want to increase or decrease the frequency that the snoop_archiver.sh runs in the crontab.

If increasing the frequency to greater than 60 minutes then you must also increase the KEEP_SECONDS variable (default 3600) in the **rolling-snoop.sh** script, otherwise when **rolling-snoop.sh** is stopped, capture files older than 60 minutes will be rolled over.

Warning: Not setting, or setting the MAX_PACKET_COUNT variable to a huge value, increases the potential for a snoop capture file to completely fill up the used space of the output disk partition to 100%.

Using External Tools for Monitoring

Introduction

This topic describes how to monitor Oracle Communications Network Charging and Control (NCC) using external monitoring tools. You can configure them to provide a real-time operational view of NCC and also helps you monitor the status of all the three components (SMS, SLC, and VWS).

In this topic, open source tools such as Pushgateway, Prometheus, and Grafana are used as an example. However, it is not restricted to only these tools. You can use any third party tool that supports the metric output.

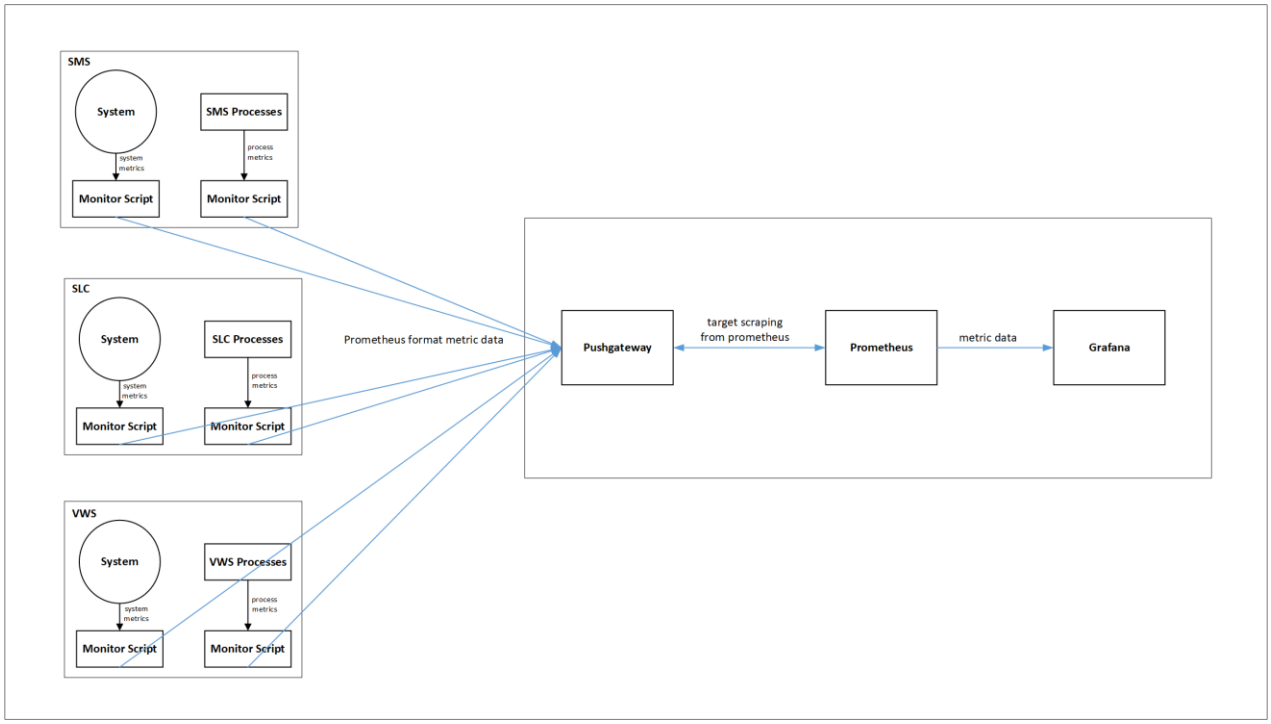
Prometheus collects and stores the metric data in time-series database and Grafana is used for graphical visualizations.

Prometheus collects the following application metrics:

- CPU Utilization

- Memory Utilization
- SLEE Resource Usage
- Process-wise Memory Utilization

Architecture Diagram



The functions of the various components are described below:

- Monitoring Scripts: Collect metrics, transform, and post to Pushgateway.
- Pushgateway: Serves as scraping target to Prometheus.
- Prometheus: Stores the metric data in time-series database.
- Grafana: Uses Prometheus data for graphical visualization and alerts.

Monitoring Scripts

For collecting metric data, transforming them into Prometheus format, and posting them to Pushgateway, the following scripts are used:

- start_system_monitor.py
- stop_system_monitor.py
- start_service_monitor.py
- stop_service_monitor.py
- start_memory_monitor.py
- stop_memory_monitor.py
- start_SLEE_resource_monitor.py
- stop_SLEE_resource_monitor.py

All the monitoring scripts are available in **/IN/service_packages/MONITORING/bin** directory.

Use the following command to run start scripts in background, in all the three application nodes (SMS, SLC, and VWS):

```
nohup <start_script_name> &
```

start_system_monitor.py

This script collects overall CPU and memory usage metric.

The Prometheus format metric generated with this script is as follows:

```
[<metric_name>{resource="CPU Usage"} <value is percentage>\n',
'<metric_name>{resource="Total Physical Memory"} <value in megabytes>\n',
'<metric_name>{resource="Free Memory"} <value in megabytes>\n']
```

Example

```
['ocncc_SLC_system_details{resource="CPU Usage"} 72.7\n',
'ocncc_SLC_system_details{resource="Total Physical Memory"} 14761\n',
'ocncc_SLC_system_details{resource="Free Memory"} 12786\n']
```

stop_system_monitor.py

This script is used to stop the system monitoring. The **start_system_monitor.py** script writes its pid into a file at **/IN/service_packages/MONITORING/tmp**, which is used by this script to stop the **start_system_monitor.py** script.

start_service_monitor.py

This script collects the details about running smf_oper processes in the node.

The Prometheus format metric generated with this script is as follows:

```
['<metric_name>{process=<process1>} <number of instances of process1 currently
running>\n', '<metric_name>{process=<process2>} <number of instances of process1
currently running>\n', ..... \n']
```

Example

```
['ocncc_SLC_service_status{process="slee_acs"} 0\n',
'ocncc_SLC_service_status{process="xmsTrigger"} 0\n',
'ocncc_SLC_service_status{process="diameterControlAgent"} 0\n',
'ocncc_SLC_service_status{process="diameterBeClient"} 0\n',
'ocncc_SLC_service_status{process="capgw"} 0\n',
'ocncc_SLC_service_status{process="ussdgw"} 0\n']
```

stop_service_monitor.py

This script is used to stop the service monitoring. The **start_service_monitor.py** script writes its pid into a file at **/IN/service_packages/MONITORING/tmp,** which is used by this script to stop the **start_service_monitor.py** script.

start_memory_monitor.py

This script collects the memory usage of smf_oper processes in the node.

The Prometheus format metric generated with this script is as follows:

```
['<metric_name>{process=<process1>, } <process1 memory usage in
kilobytes>\n','<metric_name>{process=<processX>, } <processX memory usage in
kilobytes>\n', ..... ]
```

When no processes are running, then the Prometheus format metric generated is as follows:

```
['<metric_name>{process="Service Down"} 0\n']
```

Example

```
['ocncc_SLC_service_memory_usage{process="slee_acs.14179"} 201044\n',
'ocncc_SLC_service_memory_usage{process="ussdgw.14182"} 35044\n',
'ocncc_SLC_service_memory_usage{process="capgw.14184"} 20592\n',
'ocncc_SLC_service_memory_usage{process="xmsTrigger.14196"} 58184\n',
'ocncc_SLC_service_memory_usage{process="diameterControlAgent.14213"} 23052\n',
'ocncc_SLC_service_memory_usage{process="diameterBeClient.14218"} 20724\n']
```

stop_memory_monitor.py

This script is used to stop the memory monitoring. The **start_memory_monitor.py** script writes its pid into a file at **/IN/service_packages/MONITORING/tmp**, which is used by this script to stop the **start_memory_monitor.py** script.

start_SLEE_resource_monitor.py

This script collects the following SLEE resources in the system:

- Max Dialogs
- Used Dialogs
- Max Events
- Used Events
- Max Calls
- Used Calls

The Prometheus format metric generated with this script is as following:

```
['<metric_name>{resource="Max Dialogs", } <number of total dialogs at max>\n',
'<metric_name>{resource="Used Dialogs", } <number of dialogs in use currently>\n',
'<metric_name>{resource="Max Events", } <number of total events at max>\n',
'<metric_name>{resource="Used Events", } <number of events in use currently>\n',
'<metric_name>{resource="Max Calls", } <number of total calls at max>\n',
'<metric_name>{resource="Used Calls", } <number of calls in use currently>\n']
```

Example

```
['ocncc_SLC_resources{resource="Max Dialogs", } 70000\n',
'ocncc_SLC_resources{resource="Used Dialogs", } 0\n',
'ocncc_SLC_resources{resource="Max Events", } 196208\n',
'ocncc_SLC_resources{resource="Used Events", } 22\n',
'ocncc_SLC_resources{resource="Max Calls", } 25000\n',
'ocncc_SLC_resources{resource="Used Calls", } 0\n']
```

stop_SLEE_resource_monitor.py

This script is used to stop the resource monitoring. The **start_SLEE_resource_monitor.py** script writes its pid into a file at **/IN/service_packages/MONITORING/tmp**, which is used by this script to stop the **start_SLEE_resource_monitor.py** script.

Note: All the monitoring scripts support the syntax and semantics of python 3.

Configuring Monitoring Scripts

You can configure the properties of monitoring scripts through **configurations.yml** file. Keep this file in **/IN/service_packages/MONITORING/etc** directory. Sample configuration files are provided in the same directory. Configuration file entries follow the standard YAML notations.

The following parameters are configured through **configurations.yml** file:

Note: All the parameters are mandatory.

pushgateway: This section is used to specify the server details hosting the Pushgateway.

- host: Fully qualified Pushgateway hostname.
- port: Listening port for the Pushgateway server.
- protocol: Protocol used for communication with Pushgateway server (http/https).

prometheus: This section provides the details of the metric that would be sent out. It has the following four sub-sections:

- service_monitoring: OCNCC processes metric detail. This section describes the properties of metric that is used to determine which how many instances of the monitored services (processes) are running in the system.
- memory_monitoring: OCNCC process memory metric detail. Configure the properties of the metric which tells about the memory usage of individual OCNCC processes.
- system_monitoring: OCNCC node CPU and physical memory usage metric detail.
- SLEE_resource_monitoring: OCNCC SLEE resource usage. This is only used in SLC and VWS nodes.

All the above sub-sections have the following parameters:

- ocncc_status_metric_name: Unique metric name. Metric names across all the nodes must be different.
- scrape_interval: Metric collection interval in seconds.
- logging: Whether to log metric collection output.
  - 0 - Disable logging
  - 1 - Enable logging

  Log files are available at **/IN/service_packages/MONITORING/tmp** directory.

ocncc-services: This section lists the OCNCC processes to be monitored. It is used by the following sub-sections:

- service_monitoring
- memory_monitoring

Setting up Pushgateway

Pushgateway serves as the target for Prometheus to scrape from. It listens on http port. All the monitor scripts push the metric data to Pushgateway.

Installing Pushgateway

To download Pushgateway, visit https://prometheus.io/download/#pushgateway.

Note: Pushgateway can also be set-up as a service or init job to run during the system start. By default, Pushgateway listens on http 9091 port.

Integrating Pushgateway with Monitoring Scripts

For integrating the monitoring scripts to send the metric data to Pushgateway, configure Pushgateway hostname & port details under pushgateway section in **configuration.yml** file.

Setting up Prometheus

To download Prometheus, visit https://prometheus.io/download/.

For instructions on how to install Prometheus, visit Prometheus website.

Note: Sample **prometheus.yml** config file is available in SMS node at **/IN/service_packages/MONITORING/etc/Prometheus-Grafana-Samples** directory.

Accessing Prometheus Web UI

You can access the Prometheus UI on 9090 (default port) of the Prometheus server.

Error! Hyperlink reference not valid.

Checking Targets in Prometheus UI

To check targets, access Prometheus UI and click **Status > Targets**.

It shows the targets it is scraping on (based on scrape_configs in **prometheus.yml** file).

Setting up Grafana

Installing Grafana

To download and install Grafana, visit https://grafana.com/.

Configuring Grafana

Default configurations are enough for basic use. However, if you need to change any specific parameter, refer the configuration details at https://grafana.com/docs/grafana/latest/administration/configuration/.

Accessing Grafana Web UI

You can access Grafana web UI using the following URL:

http://*hostname*:3000

where *hostname* is the name of the host where Grafana is running.

By default, Grafana listens on port 3000. Details about the default credentials are available in official Grafana documentation.

Creating Data Source

For instructions on creating data source, visit https://grafana.com/.

Creating Dashboards

You can configure Grafana dashboards to visualize and monitor the metrics from the data source. For instructions on creating dashboards, visit https://grafana.com/.

Note: Sample JSON files for dashboards are available in SMS node at **/IN/service_packages/MONITORING/etc/Prometheus-Grafana-Samples** directory.
You can import and edit it as per the requirement.

Configuring Alerts in Grafana

You can configure alerts in Grafana for all the metrics being exported from OCNCC.
For information about creating alerts, visit https://grafana.com/docs/grafana/latest/alerting/.

Monitoring SIGTRAN Traffic with Prometheus and Grafana

You can monitor SIGTRAN traffic in real time using Prometheus and Grafana. Dashboards display current CAP messages, transactions per second (TPS), and transaction latency, providing operational health and performance tracking.

The SIGTRAN application collects metrics and pushes them to a configured Prometheus PushGateway endpoint.

Prometheus scrapes this data at regular intervals, and Grafana visualizes these metrics. Metrics remain available even if the application or node restarts.

The following metrics are collected:

Name	Type	Description	Labels	Details/Buckets
<moduleName>_caps_total	Counter	Total number of calls received.	`direction="ingress", event="CAPs", node, process`	Increments on each CAP event
<moduleName>_tps_total	Counter	Total number of transactions received.	`direction="ingress", event="TPS", node, process`	Increments on each TPS event
<moduleName>_milliseconds	Histogram	End-to-end latency of CAP transactions	`direction="ingress", event="latency", node, process`	Buckets: "1,5,10,20,50,100,200,500,1000,2000,5000" milliseconds

How to Enable Monitoring

To enable SIGTRAN monitoring, follow these steps:

Note: It is assumed that the Pushgateway host and port are already configured.

1 Open the ESERV config file used in m3ua process for editing

2 In the **METRICS** section, add or update a block for your subsystem (e.g., M3UA). Set all the required parameters as shown in the example below.

```
```
METRICS = {
pushGatewayHost = "YOUR_PUSHGATEWAY_HOSTNAME" # Required: Hostname or IP for PushGateway
pushGatewayPort = "9091" # Required: PushGateway port
```
```

```
M3UA = {
moduleName = "sigtran"      # Required: Metric name prefix
pushGatewayJob = "sigtran_metrics" # Required: Job label/group in
Prometheus
nodeLabel = "slc"    # Required: Node/site/instance identifier label
processLabel   = "m3ua_if"  # Required: Process/service identifier label
enableMetrics  = "true" # Set "true" to enable monitoring for this
process/subsystem (default: false)
latencyBuckets = "150,200,500"  # Optional: Histogram buckets in ms
(default: 1,5,10,20,50,100,200,500,1000,2000,5000)
pushInterval   = 1000     # Optional: Push interval in ms (default: 5000)
}
}
```

3 Save the configuration file.
4 Restart your application or the relevant subsystem for changes to take effect.

How to Disable Metric Collection

To disable metric collection for a subsystem, either:

- Set **enableMetrics = "false",** or
- Omit the **enableMetrics** parameter entirely.

No metrics will be collected or pushed for that subsystem. The application log confirms metric reporting is disabled.

Parameter Requirements and Effects

- Metrics will only be collected and pushed if:
  - **enableMetrics** is set to true, and
  - All other required parameters (pushGatewayHost, pushGatewayPort, moduleName, pushGatewayJob, nodeLabel, processLabel) are present and valid.
- If any required parameter is missing/invalid, or if **enableMetrics** is set to false or omitted, no metrics will be collected or pushed for that process/subsystem; the application logs will state the reason.
- Optional parameters (latencyBuckets, pushInterval) will use default values if not specified.
- After making configuration changes, restart the application/process to apply the new monitoring settings.

Troubleshooting

If you do not see metrics in Prometheus or Grafana:

- Confirm all required parameters are set and correct.
- Ensure the PushGateway is reachable from the application node.
- Check application logs for any warnings or errors about metrics export or configuration.

Example Dashboard Queries

- **Total CAPs:**
  sum(sigtran_caps_total{direction="ingress"})
- **Total TPS:**

sum(sigtran_tps_total{direction="ingress"})

- **99th percentile Latency:**
  histogram_quantile(0.99, sum(rate(sigtran_latency_milliseconds_bucket[5m])) by (le))
- **1-min average Latency:**
  rate(sigtran_latency_milliseconds_sum[1m]) / rate(sigtran_latency_milliseconds_count[1m])

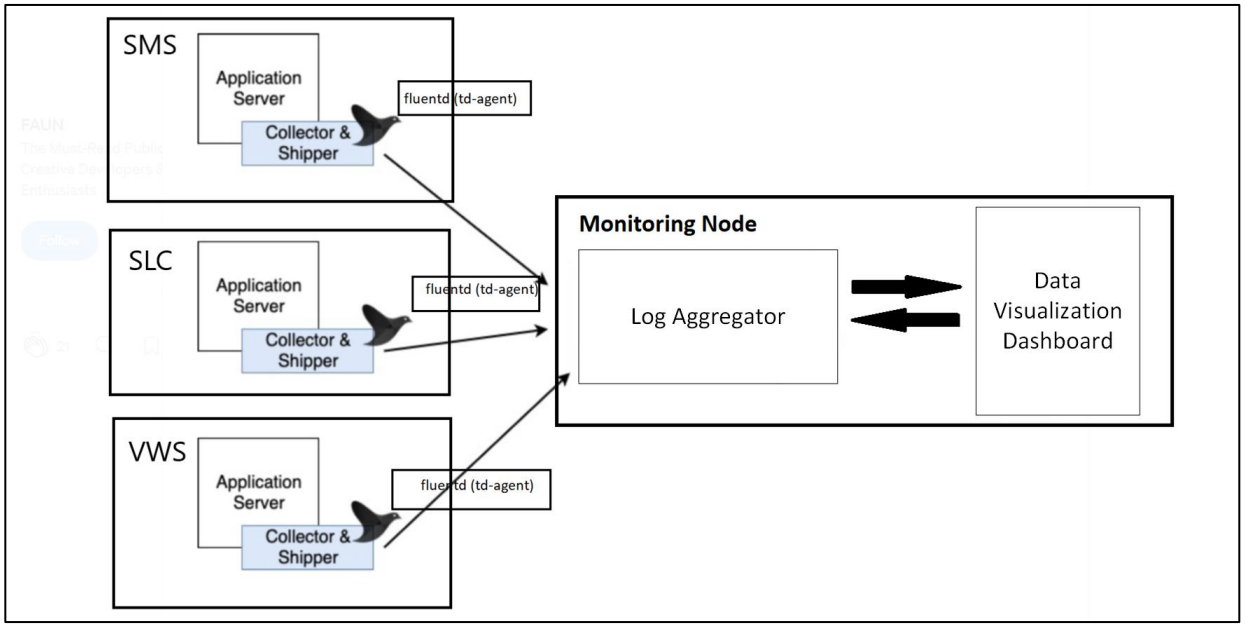
Using External Tools for Logging

Introduction

You can use log aggregators and data visualization dashboards for logging services in NCC. For example, you can use open-source tool such as Fluentd for collecting, transforming and shipping log data to the log aggregator. Fluentd is a data collector on the nodes to tail the log files, filter and transform the log data, and deliver it to the log aggregator, where it is indexed and stored. You can use any log aggregator and visualization dashboard to monitor logs.

This setup helps you to have a real-time operational view of NCC. You can integrate this with the application to visualize all the application logs through the data visualization dashboard. You can also configure it to notify or trigger alerts whenever a specified error message occurs in the logs.

Architecture Diagram



Installing Fluentd

If you are using Fluentd as the data collector, perform the following steps after the installation:

5 Install **fluent-plugin-concat** for merging multiple logstash. To install the plugin, run the following command:

```
/usr/sbin/td-agent-gem install fluent-plugin-concat
```

6 To add td-agent to esg group, run the following command as a root user:

```
usermod -G esg td-agent
```

7 To check if td-agent is part of esg, run the following command:

```
id td-agent
```

8 Create three different index for three machines (SMS, SLC, and VWS) to display the logs in their respective index.

To create a new index, run the following commands (where the log aggregator and the data visualization tools are running):

```
curl -XPUT http://localhost:9200/sms_index
curl -XGET http://localhost:9200/sms_index
curl -XPUT http://localhost:9200/slc_index
curl -XGET http://localhost:9200/slc_index
curl -XPUT http://localhost:9200/vws_index
curl -XGET http://localhost:9200/vws_index
```

9 To delete your index, run the following command:

```
curl -XDELETE http://localhost:9200/sms_index
```

10 To display the index in a more structured view, run the following command:

```
curl -XGET http://localhost:9200/sms_index?pretty
```

11 Edit the **/etc/td-agent/td-agent.conf** file to define the required Fluentd plugins and patterns, so that Fluentd collects the logs from the source path and ships it to the log aggregator.

Sample **td-agent.conf** file path in SMS: **/IN/service_packages/MONITORING/etc/**

Note: After editing **td-agent.conf** file, restart td-agent.service for the changes to take effect.

Chapter 4

Service Logic Controller (SLC)

Overview

Introduction

This chapter explains how and why the SLC is used.

In this chapter

This chapter contains the following topics.

Service Logic Controller Overview

Introduction

The SLC is used to handle calls using compiled control plan logic which is initially defined on the SMS and replicated to each SLC node.

All the main processing takes place inside the SLEE. On the SLC, this processing is primarily handled by slee_acs. Depending on the protocols involved on the network, a number of Network Connectivity Agent processes will also be running.

Service Logic Execution Environment

Introduction

The main process working to handle requests between the network and the SLEE is slee_acs. slee_acs is part of the Oracle Communications Network Charging and Control (NCC), and is located in **/IN/service_packages/ACS/**

For more information on configuration options, refer to the appropriate product technical guide.

SLEE.cfg

Configuration for the SLEE in general is contained in **/IN/service_packages/SLEE/etc/SLEE.cfg**. This contains configuration for the resources allocated to the SLEE, and the applications, services and servicekeys of applications running on the SLEE.

- MAX<Parameter>: Contains the maximum resources allocated to the SLEE, such as
  - Applications
  - Services
  - Dialogs

  - Events
  - Calls
- APPLICATION: Contains the location of the application startup scripts, and how many instances to run.
- INTERFACE: Contains the definitions for interfaces on the SLEE, their startup scripts and interface type.
- SERVICE: Defines the applications/interfaces as a service (can be done multiple times for each application/interface).
- SERVICEKEY: Defines the service to be triggered for received service keys.

ACS.conf

Configuration for other ACS components are contained in **/IN/service_packages/ACS/etc/acs.conf**. This contains general configuration options for the following processes:

- acsStatisticsDBInserter
- acsStatsMaster
- acsStatsLocal
- acsCompilerDaemon

It also contains configuration for acsChassis, which specifies:

- Plug-ins
- Services
- Normalization
- ACS EDR generation
- Some other specific call-handling scenario configuration options

eserv.config

Remaining configuration is primarily found inside **/IN/service_packages/eserv.config**. Each product has a top level section, (for example "ACS {}" for ACS) and the underlying processes for each product are configured in sub-sections of **eserv.config**.

There are a few exceptions to this. Most notably SUA and M3UA Interface configuration, which is found in **/IN/service_packages/SLEE/etc/**

SLEE Watchdog

The SLEE watchdog is responsible for keeping track of all the processes running inside the SLEE. Upon SLEE startup, all the processes are registered with the watchdog. The watchdog periodically checks each process to make sure it is processing events correctly.

If not, the watchdog marks the process as suspect and sends the process a management event. During the next watchdog cycle (default 30 seconds) the watchdog will check that the event has been processed. If the event was not actioned, the process will be aborted and restarted.

Abort information

Whenever a process is terminated or restarted by the watchdog, there are appropriate records in **SLEE.log/syslog**, for example:

```
watchdog(18186) WARNING: Interface beVWARS3 does not exist at PID 18169, presuming dead.
Jul 18 00:53:16.091448 watchdog(18186) WARNING: Sending SIGABRT to interface beVWARS3, process
18169.
Jul 18 00:53:42.250416 watchdog(18186) WARNING: Interface beVWARS3 does not exist at PID 18169,
presuming dead.
Jul 18 00:53:42.250844 watchdog(18186) WARNING: Restarting interface beVWARS3 (was process
18169).
```

```
Jul 18 00:53:42 watchdog(18186) SleeInterfaceInstance::start()=3 sleeInterfaceInstance.cc@156:
Starting Interface beVWARS3.sh: PID: 8237
```

The watchdog also has built in deadlock prevention. A timer is set before beginning a check loop to ensure that it does not get deadlocked on a semaphore. If the timer expires, the watchdog believes the SLEE is having serious issues and will restart the entire SLEE.

Note: Some interactions with a process can stop it from responding to watchdog management events, one common example of this is a gcore. In this situation, the watchdog should be killed or sent a SIGUSR1 signal to stop operating.

The SLEE will need to be restarted in order for the watchdog to become operational again.

Update loader

The updateLoader process on the SLC is the client-side of IN Replication. It is responsible for receiving and applying updates from the smsMaster process on the SMS.

The updateLoader is run from inittab, and runs in run-level 2.

While not a traffic handling application, this process is absolutely crucial to the heath of the platform; non-replicated changes can cause major subscriber issues and revenue loss on a solution.

Checking replication status

On the SLC side, the updateLoader process and log file can be checked to make sure the process is up and running, and not presenting any errors.

See the SMS Replication section for information on the SQL queries required to check replication directly from the SMS database.

Full replication

In the event of a replication issue, it may be required to instruct the updateLoader to perform a full resync.

This performs a full collection of data from the SMS database. Depending on the amount of data, and what information is configured to replicate (for example, replicating subscriber data will mean information for every subscriber on the platform is replicated) replication can take a number of hours to complete.

Performing a Full Resync

To perform a full resync, open the updateLoader startup script, and add the -resync argument to the command line (highlighted), and restart the updateLoader process. Enter:

```
$ vi /IN/service_packages/SMS/bin/updateLoaderStartup.sh
```

Edit the result as shown:

```
. /IN/service_packages/SMS/.profile-scp
echo "`date` - Waiting for DB SCP"
pid=""
while [ -z "$pid" ] ; do
    pid=`ps -ef|grep "ora_pmon_SCP"|awk '$8!="grep"{print$2}'`
    if [ -z "$pid" ] ; then
        echo "..."
        sleep 30
    fi
done
echo "`date` - DB SCP is ready"
echo "`date` - Waiting for Replication Cfg"
while [ ! -f "/IN/service_packages/SMS/etc/replication.config" ] ; do
    echo "..."
    sleep 30
```

```
done
echo "`date` - Replication Cfg is ready"
exec /IN/service_packages/SMS/bin/updateLoader -nodeid 301 -resync
```

Enter:

```
$ pkill updateLoader
```

Result: The updateLoader is restarted in full resync mode.

Resync progress

The progress of a Full Resync can be monitored through **/IN/service_packages/SMS/tmp/updateLoader.log**:

```
Thu Nov 11 01:09:41 GMT 2010 - Waiting for DB SCP
Thu Nov 11 01:09:41 GMT 2010 - DB SCP is ready
Thu Nov 11 01:09:41 GMT 2010 - Waiting for Replication Cfg
Thu Nov 11 01:09:41 GMT 2010 - Replication Cfg is ready
initialiseNode: Reading '/IN/service_packages/SMS/etc/replication.def'
…

Node 301 SMS comparison/resync client ready.
Oct 11 01:09:44.966814 updateLoader(21378) NOTICE: Update Loader replication process started
(node 301)
Cancelling any current client action.
Oct 11 01:09:44.979322 updateLoader(21378) NOTICE: Reached master node 1 at `<IP Address>'
RES: Thu Oct 11 01:10:08 2010: Node 301, started processing X SMS and Y SCP records.
RES: Thu Oct 11 01:10:08 2010: Node 301, resynchronisation pass 1, started processing of X SMS
and Y SCP records.
Oct 11 01:10:08.291659 smsCompareResyncClient(21515) NOTICE: Beginning resynchronisation for
node 301.
RES: Thu Oct 11 01:10:18 2010: Node 301, table NP_DN_RANGE, group NP_DN_RANGE_0, processed A of
X SMS and B of Y SCP records.
Nov 11 01:12:05.308062 updateLoader(21378) NOTICE: Resynchronization Finished.  Processing
Queued Updates
Node 301 SMS comparison/resync client ready.
Nov 11 01:12:05.353114 updateLoader(21378) NOTICE: Finished Processing Queued Updates
```

The process will periodically report how far through the resync it is, including number of rows (out of total <node type> rows - highlighted example).

Once complete, updateLoader will return to regular operations. It is recommended to remove the -resync flag as soon as the resync has finished and restart the process.

Network Connectivity Agents

Introduction

Network connectivity agents (NCAs) exist to interface the various protocols running on the network with slee_acs and the rest of the SLEE.

For most NCAs, there is an associated process running to translate the incoming protocol to the internal protocol used by slee_acs (INAP).

Example NCAs

This table lists some examples of NCAs:

Protocol	Package	Process	Log File Location
Diameter (Northbound)	DCD	diameterBeClient	/IN/service_packages/DCD/tmp
Diameter (Southbound)	DCA	diameterControlAgent	/IN/service_packages/DCA/tmp
MAP	MMX	xmsTrigger (via adapter)	/IN/service_packages/XMS/tmp
SIP	SCA	sca_if	/IN/service_packages/SLEE/tmp

Protocol	Package	Process	Log File Location
SMPP	MMX	xmsTrigger (via adapter)	/IN/service_packages/XMS/tmp
SIGTRAN	SIGTRAN	sua_if / m3ua_if	/IN/service_packages/SLEE/tmp
XML/TCAP	TCAP_IF	xmlTcapInterface	/IN/service_packages/SLEE/tmp
USSD	UIS	ussd_gw	/IN/service_packages/UIS/tmp
SOAP/XML	OSD	osdInterface	/IN/service_packages/OSD/tmp
IS41	IS41	cdmagw (m3ua/sua)	/IN/service_packages/IDS41/tmp

Information logging

Each NCA generally writes to its own log file and to the syslog. This should help identify what the program was doing right before experiencing an issue. This information is a key requirement when raising SRs with Oracle support.

Checking Services

Introduction

The SLEE service on the SLC can be checked using the interactive slee-ctrl interface. Slee-ctrl is part of the slee-ctrl package, and is generally available on all machines that run a SLEE.

Note: The *Monitoring and Managing* (on page 19) chapter covers checking of services in more detail.

Interactive interface

To enter the interactive interface, run slee-ctrl with no arguments. Enter:

```
$ slee-ctrl
```

Result: You are then presented with some environment information, and a slee-ctrl prompt:

```
SLEE Control: v,1.0.11: script v,1.21: functions v,1.48: pslist v,1.118

User acs_oper; session [5570]; terminal pts/3; started Thu Oct 28 03:53:48 GMT 2010

The following variables are set and will be used to run the SLEE.
 SLEE_USER    acs_oper
 SLEE_SCRIPT  /IN/service_packages/SLEE/bin/slee.sh
 SLEE_CONFIG  /IN/service_packages/SLEE/etc/SLEE.cfg
 SLEE_LOG     /IN/service_packages/SLEE/tmp/SLEE.log
[acs_oper] slee-ctrl>
```

Tip: Entering **`help`** at prompt lists the valid commands.

Example status reporting

From the prompt, you can issue a series of different commands to interact with the SLEE, or get information about resources, and other things.

Some examples are shown below:

Status

To check the current status of the SLEE, including processes, enter:

```
[acs_oper] slee-ctrl> status
----------------------- Thu Oct 25 03:56:23 GMT 2010 -------------------------
```

```
C APP  USER      PID PPID    STIME COMMAND
6 SLEE acs_oper 1402    1 00:12:35 /IN/service_packages/ACS/bin/slee_acs
1 SLEE acs_oper 1404    1 00:12:35 /IN/service_packages/SLEE/bin/capgw
1 SLEE acs_oper 1405    1 00:12:35 /IN/service_packages/RAP/bin/rap
1 SLEE acs_oper 1406    1 00:12:35 /IN/service_packages/LCP/bin/locApp
1 SLEE acs_oper 1407    1 00:12:35 /IN/service_packages/ACSUSC/bin/slee_acs
1 SLEE acs_oper 1408    1 00:12:35 /IN/service_packages/UIS/bin/ussdgw
1 SLEE acs_oper 1409    1 00:12:35 /IN/service_packages/SLEE/bin/timerIF
1 SLEE acs_oper 1410    1 00:12:35 /IN/service_packages/SLEE/bin/alarmIF
1 SLEE acs_oper 1411    1 00:12:35 IN/service_packages/ACS/bin/acsStatsLocalSLEE
1 SLEE acs_oper 1412    1 00:12:35 /IN/service_packages/SLEE/bin/replicationIF
1 SLEE acs_oper 1413    1 00:12:35 /IN/service_packages/E2BE/bin/BeClient
1 SLEE acs_oper 1414    1 00:12:35 /IN/service_packages/OSD/bin/osdInterface
1 SLEE acs_oper 1415    1 00:12:35 /IN/service_packages/SCA/bin/sca
1 SLEE acs_oper 1416    1 00:12:35 /IN/service_packages/RIMS/bin/rims
1 SLEE acs_oper 1417    1 00:12:35 /IN/service_packages/XMS/bin/xmsTrigger
4 SLEE acs_oper 1419    1 00:12:35 /IN/service_packages/SLEE/bin/m3ua_if
1 SLEE acs_oper 1427    1 00:12:36 /IN/service_packages/SLEE/bin/mapGenIF
1 SLEE acs_oper 1430    1 00:12:36 IN/service_packages/SLEE/bin/xmlTcapInterface
1 SLEE acs_oper 1432    1 00:12:36 /IN/service_packages/SLEE/bin//watchdog
total processes found = 27 [ 27 expected ]
================================= run-level 3 ==================================
```

Resources

To check the status of SLEE resources, in particular memory/CPU usage, enter:

```
[acs_oper] slee-ctrl> resources
```

```
----------------------- Fri Oct 29 01:28:31 GMT 2010 -------------------------
APP  USER       PID PPID S %CPU %MEM    VSZ    RSS  TIME  ELAPSED COMMAND
SLEE acs_oper 10267    1 S  0.0  4.0 633584 314720 00:02 04:45:10 slee_acs
SLEE acs_oper 10268    1 S  0.0  4.0 633584 314720 00:02 04:45:10 slee_acs
SLEE acs_oper 10269    1 S  0.0  4.0 633584 314728 00:02 04:45:10 slee_acs
SLEE acs_oper 10270    1 S  0.0  4.0 633584 314712 00:02 04:45:10 slee_acs
SLEE acs_oper 10271    1 S  0.0  4.0 633584 314728 00:02 04:45:10 slee_acs
SLEE acs_oper 10272    1 S  0.0  4.0 633584 314720 00:02 04:45:10 slee_acs
SLEE acs_oper 10273    1 S  0.1  3.4 527728 271768 00:43 04:45:10 timerIF
SLEE acs_oper 10274    1 S  0.0  3.4 527760 271800 00:00 04:45:10 alarmIF
SLEE acs_oper 10275    1 S  0.0  3.4 528040 272296 00:07 04:45:10 acsStatsLocalSLEE
SLEE acs_oper 10276    1 S  0.1  3.5 539712 276120 00:44 04:45:10 replicationIF
SLEE acs_oper 10277    1 S  0.1  3.7 552960 290896 00:53 04:45:10 BeClient
SLEE acs_oper 10278    1 S  0.0  3.6 544856 281864 00:00 04:45:10 xmlTcapInterface
SLEE acs_oper 10279    1 S  0.1  3.4 527784 271824 01:08 04:45:10 watchdog
total processes found = 13
================================= run-level 3 ==================================
Memory: total 7.9G, used 6.1G (77.6%) + 128.0K (0.0%) /tmp, free 1.8G (22.4%) OK
Swap: total 4.4G, used 2.1G (48.9%), free 2.2G (51.1%) OK
```

Call resources

To check the status of SLEE call resources, in particular free calls and events, enter:

```
[acs_oper] slee-ctrl> check 1
```

```
SLEE: Using shared memory offset: 0xc0000000
01:29:03        Dialogs Apps    AppIns  Servs   Events   Calls
                [70000] [30]    [296]   [30]    [207152] [25000]
01:29:03        70000   29      290     24      207146    25000
01:29:04        70000   29      290     24      207146    25000
01:29:05        70000   29      290     24      207146    25000
01:29:06        70000   29      290     24      207146    25000
01:29:07        70000   29      290     24      207146    25000
01:29:08        70000   29      290     24      207146    25000
01:29:09        70000   29      290     24      207146    25000
```

A dwindling number of free calls indicates a call leak (this means the number of available calls that the SLC is decreasing in such a manner that eventually it will no longer be able to serve traffic). In this situation the only real solution is to restart the SLEE and reset the available resources.

If the problem continues to happen, it will be prudent to investigate the type of traffic hitting the platform and attempt to determine what is triggering the leak. Slower response times from other network elements can also cause a decrease in free SLEE resources.

Stop and start

To stop, start, or restart the SLEE, enter as required at the prompt:

```
[acs_oper] slee-ctrl> stop
[acs_oper] slee-ctrl> start
[acs_oper] slee-ctrl> restart
```

Handling Database Connection Reset

If the database connection on SLC node is reset, restart the SLEE processes on it. A full resync has to been done before restarting the SLEE processes.
When the database goes down, or the connection gets reset on SLC, slee_acs is not automatically restarted because of the following reasons:

- For all of the active calls in SLC, the required session data such as call context, control plan, and subscriber details etc. are cached during the call initiation. If the SLEE processes are restarted because of database reconnection, all the ongoing session data are lost, and all of the ongoing calls will end abruptly. In order to avoid this, all the processes are kept running to be able to serve all the subsequent request of active calls.
- When database goes down, a full resync is required to get the database synched for any profile updates and other database changes. Reconnection to database, without the resync would cause a lot of inconsistent data and configuration related issues.

In such scenarios, any new call will fail because slee_acs will not be able to fetch initial configuration data for the subscriber.

Chapter 5

Service Management System (SMS)

Overview

Introduction

This chapter explains how and why the SMS is used.

In this chapter

This chapter contains the following topics.

Service Management System Overview

Introduction

The SMS is responsible for setup and maintenance of many aspects of the platform and contains the web-based Java front-end for viewing and altering configuration.

Behind the scenes, the SMS receives EDRs from the SLC and VWS and processes them before archival, removal, or forwarding for further processing by external systems.

The SMS also runs a number of additional services, including the Provisioning Interface (PI), the alarm, statistics and replication subsystems, along with processes for billing interaction and Business Processing Logic (BPL).

Java Screens

Introduction

The front end for configuring the NCC runs through Java on the SMS. To access, please start **smsGui.bat/smsGui.sh**.

Oracle Listener

Connections to the database are handled by the Oracle Listener. If you are experiencing problems connecting to the Java screens, check the status of the listener by using lsnrctl.

To check the status of the listener, at the $ prompt, enter the following command as the oracle user:

```
$ lsnrctl status
```

The following example is for the Oracle 11*g* database when the listener is configured correctly. It shows the listener running and the handlers for the service. You would see similar output reported for the Oracle 12*c* database when the listener is configured correctly.

```
LSNRCTL for Linux: Version 19.0.0.0.0 - Production on 05-MAR-2025 13:34:53

Copyright (c) 1991, 2024, Oracle. All rights reserved.

Connecting to (ADDRESS=(PROTOCOL=tcp)(HOST=)(PORT=1521))
STATUS of the LISTENER
------------------------
Alias                     LISTENER
Version                   TNSLSNR for Linux: Version 19.0.0.0.0 - Production
Start Date                05-MAR-2025 06:04:33
Uptime                    0 days 7 hr. 30 min. 20 sec
Trace Level               off
Security                  ON: Local OS Authentication
SNMP                      OFF
Listener Log File /u01/app/oracle/product/19.0.0/network/log/listener.log
Listening Endpoints Summary...
  (DESCRIPTION=(ADDRESS=(PROTOCOL=tcp)(HOST=host)(PORT=1521)))
Services Summary...
Service "SMF.host.SMF" has 1 instance(s).
  Instance "SMF", status READY, has 1 handler(s) for this service...
The command completed successfully
```

If there are problems connecting to the Java screens because the listener is configured incorrectly, then the lsnrctl status output will appear differently; for example, the output could show that the listening port or hostname is incorrect, or that the listener is not running.

Note: If you encounter a failure message, contact your system administrator.

Starting and stopping Listener

You can start and stop the Listener by using the lsnrctl start command and the stop command. Enter at the $ prompt:

```
$ lsnrctl start
```

```
$ lsnrctl stop
```

Listener configuration

Configuration for the Listener is in the file **$ORACLE_HOME/network/admin/listener.ora**. Generally, this information will not need to change, unless an aspect (for example, hostname, or IP resolution) of the network changes.

Java packages

The front-end Java packages are contained in **/IN/html/smsGui.bat** and **/IN/html/smsGui.sh** calls **SMS.sig.jar**, which makes up the foundations of the screens.

Additional packages exist for the various products installed: for example, ACS (**acs.sig.jar**), CCS (**ccs.sig.jar**), and so on.

Control Plan Editor

The Control Plan Editor (CPE) is a front end for designing call handling logic. In general, every single event that triggers the NCC platform will hit a control plan at some point.

The control plan used and nodes travelled by an event triggering the platform are written to the associated ACS EDR in the CPN and TFN tags.

Each event will have an Event Detail Record (EDR), the only real exception being when an engine error occurs and processing cannot continue. In this scenario, ACS would still write a call dump, so a record should still be available.

Tracing a Control Plan

If a problem can be traced to an ACS EDR, the CPE can be used to find the control plan used and to trace through each node to see exactly how the event traversed it.

Follow these steps to trace a control plan through the CPE.

Step	Action
1	From the main SMS menu select **Services -> ACS Service -> Control Plans**.
2	Select the appropriate customer from the top right-hand **Customer** box.
3	Select **Open.**
4	Select the appropriate control plan, based on the **CPN** tag.
5	Press **N** to show node numbers.
6	Using the contents of the **TFN** tag, follow the calls progress through the plan.

Tips:

- Clicking one time on a node will highlight and indicate all connected nodes and associated exits, along with tool-tip descriptions. Note that the connection highlighting does not occur if the Control Plan is opened with the structure read-only.
- Sub-Control Plan nodes (SCPN) can be opened in a **view** mode directly from the node edit screen.
- The Start and End node IDs are not reported when a SCPN node is triggered; as a result, it can sometimes be difficult to trace how a Sub-Control Plan ended.

acsCompilerDaemon

When a control plan is saved, the dialog box shown on screen is the output of the SMS process acsCompilerDaemon, which validates and compiles control plans.

If there is an issue compiling a control plan, and the CPE Save dialog box does not contain enough information to go forward, consider putting acsCompilerDaemon into DEBUG to get more information about why the compilation failed.

Tip: If the save, dialog box is completely blank, acsCompilerDaemon may not be running at all.

ccsBeOrb

When the CCS portion of the screens need to interact with the VWSs, this is done through ccsBeOrb.

The most common reason for the screens needing to query the VWS is when opening the Edit Subscriber screen. This displays all real-time Wallet information about a subscriber.

If both the VWSs are restarted and there is a complete billing outage, it can take a couple of minutes for ccsBeOrb to realize it has disconnected and reconnect. This process can be sped up by restarting ccsBeOrb.

The error message below is indicative of this problem:

However, if the error message remains, there is a definite problem with the connection between the SMS and the VWSs.

Replication

Introduction

The SMS is responsible for managing replication on the platform.

From the front-end, configuration is handled through the Java screens in **Operator Functions -> Node Management**, and then:

- **Replication Nodes** configures the node number and address.
- **Table Replication** contains tables to be replicated to sub-nodes.
- **Replication Node Types** allows the configuration of node types with pre-determined tables replicated.

Configuration completed

When configuration is complete (or has changed) a special file called **replication.config** is generated and put on all replication nodes in **/IN/service_packages/ SMS/etc/**. Any processes that use the smsMaster as a means of replication use this file to decide where to send replication updates to, and what to replicate (in the case of updateLoader).

Processes include:

- Alarm subsystem (smsAlarmDaemon)
- Statistics subsystem (smsStatsDaemon)
- Data replication (updateLoader)
- Upstream replication (replicationIF)

When the SLC or VWS needs to replicate a change (usually to the SMS), this is sent upstream through replicationIF.

Checking replication status

Replication can be checked using a few queries on the database on the SMS.

There are three main tables required to check the status of replication:

12 REP_ORA_RENUMBERED
This table contains all the information that is yet to be replicated.

13 REP_PENDING_QUEUE
This table contains the ID of the last event replicated.

14 REP_CNF_NODE
This table contains a list of the configured replication nodes for reference.

Viewing backlog

To view the current backlog of changes waiting to be replicated, run the following SQL query as smf_oper on the SMS. This command will show how many values are yet to be replicated down to the nodes. The larger the number, the further behind replication currently is. Enter:

```
SQL> select count(*), node_number from rep_ora_renumbered group by
node_number;

  COUNT(*) NODE_NUMBER
---------- -----------
       162         301
       129         302
       122         351
       116         352
   6305218
```

Event Id checking

Another way to check the current status of replication is to use the following SQL query to list the the minimum and the maximum ID. Enter:

```
SQL> select min(event_id), max(event_id) from rep_ora_renumbered

MIN(EVENT_ID)    MAX(EVENT_ID)
-------------    -------------
164431033           170738029
```

If the difference between Min and MAX is large as above, this indicates a clear problem with replication.

Problem node ID

Once you have determined that there is a problem with replication, it is important to determine if one particular node has fallen behind or if replication has completely failed. To do this, you must run the following SQL. Enter:

```
SQL> select * from rep_pending_queue;

    NODE_ID       ROE_EVENTID
    ----------    -----------
      301           164432645
      302           170738026
      351           170738026
      352           170738026
```

The ROE_EVENTID is the current event ID that the node is processing. If the event ID is close to the maximum event ID (170738029), then the node is essentially in sync, but if the ROE_EVENTID is a long way off, that node is having issues receiving updates.

Problem node name

Once it is determined which NODE_ID is behind on replication, run the following SQL command to resolve the name of the node. Enter:

```
SQL> select description, node_number from REP_CNF_NODE;

DESCRIPTION                          NODE_NUMBER
------------------------------------ -----------
UAS01                                        301
UAS02                                        302
UBE01                                        351
UBE02                                        352
```

In this example, we can see that the node_id that is behind (301) is connected to UAS01.

Problem resolution

The two most common ways to resolve a replication issue is to:

- Perform a full resync on the SLC (see *Full replication* (on page 45) for the details)
- Restart the smsMaster on the SMS.

EDR Management

Overview

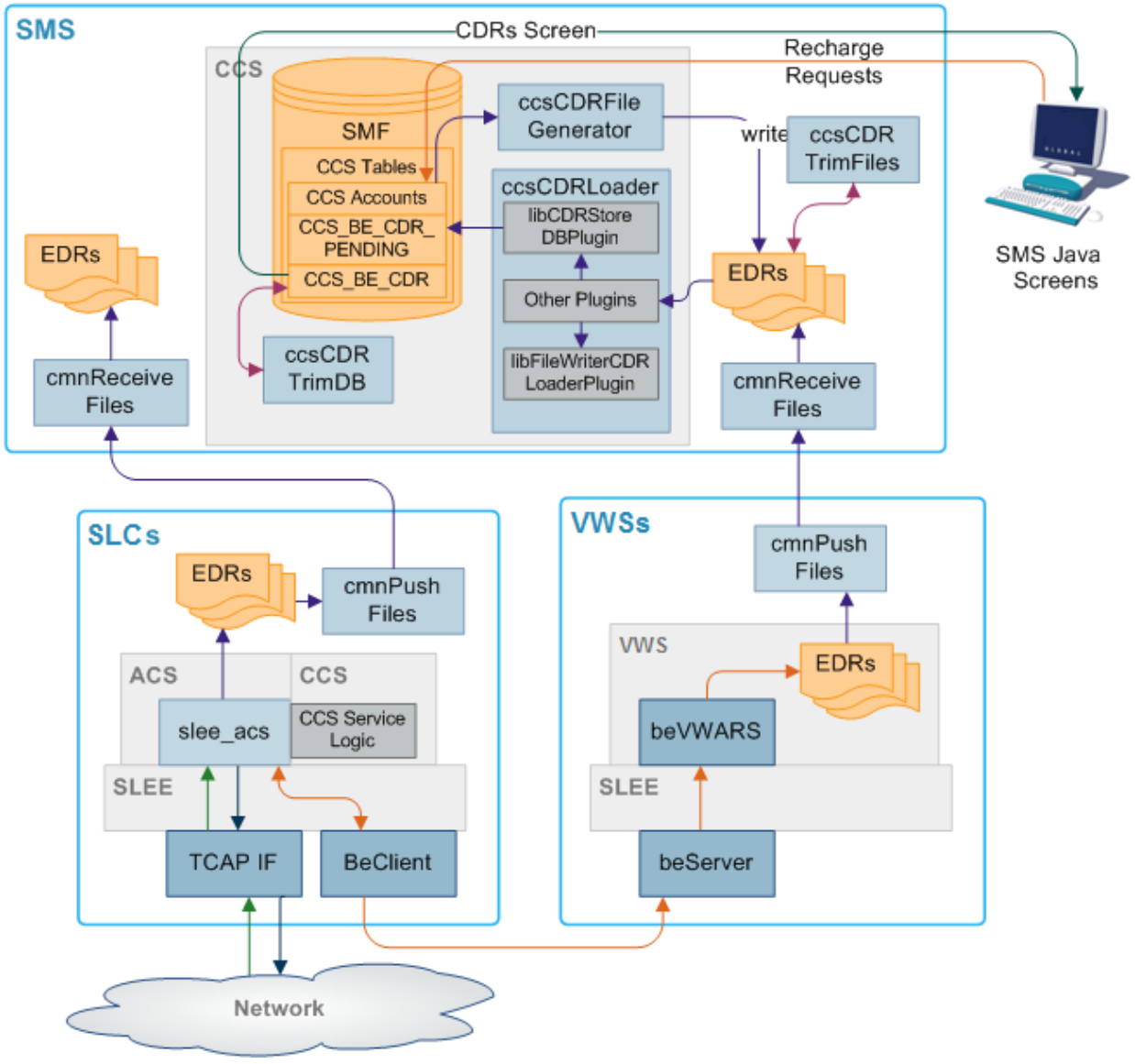
The SMS is generally responsible for receiving, processing and archiving EDRs received from the other nodes on the NCC platform.

cmnPushFiles is responsible for "pushing" EDRs to the SMS, which receives them via cnmReceiveFiles.

Depending on the location they are received to, they are processed by other processes or scripts, before being archived.

EDR process flow diagram

Here is an overview of the EDR process flow:



Receiving files

Incoming EDRs are received by the network service cmnReceiveFiles. The destination of the EDR is controlled by the cmnPushFiles process on the sending machine.

EDR are generally separated into different directories for different EDRs. For example, an ACS EDR would be sent to **/IN/service_packages/SMS/cdr/received**, and BE EDRs would be sent to **/IN/service_packages/CCS/logs/CDR-in**.

This allows for more targeted troubleshooting when looking at EDR flow.

Processing EDRs

The processing of EDRs depends on their location.

Processing for CCS or BE EDRs is handled by ccsCDRLoader, which runs from inittab as a background process.

Under normal circumstances ACS EDRs are not processed at all, and are just archived.

ccsCDRLoader

Billing EDRs originating from the VWSs are processed in real-time by ccsCDRLoader.

As soon as an EDR is found in the incoming directory, it is picked up by the ccsCDRLoader process and run through the configured plug-ins.

Configured plug-ins

Configuration, such as incoming/outgoing directory, enabled plug-ins and plug-in settings are defined in **/IN/service_packages/eserv.config**. The results of the processing are dependent on the plug-ins that are running.

Plug-ins include:

- AcsCustIdPlugin
  Ensures ACS Customer ID is present, if not it will look it up in the database
- VoucherRedeemPlugin
  Ensures voucher redemption is updated in the CCS_VOUCHER_REFERENCE table in the SMS database
- AcctHistPlugin
  Ensures the CCS Account History table is updated with subscriber changes
- CDRStoreDBPlugin
  Writes the EDR details to the CCS_BE_EDR table in the SMS database
- FileWriterCDRLoaderPlugin
  Writes the EDR details to file after ccsCDRLoader has finished processing
  If configured, can also alter the timezone of the date fields

With this in mind, if the CCS Subscriber screens show no EDR history for a subscriber that has made calls, the ccsCDRLoader would be one place to check for issues.

Archiving

Similar to processing, the archiving of EDRs is dependent on the location of the files. After processing, billing EDRs are moved to a final location, for example, **/IN/service_packages/CCS/logs/CDR-store**.

The archival of EDRs is done by three scripts:

1 ACS EDR
  /IN/service_packages/SMS/bin/smsCdrProcess.sh
2 BE EDR
  /IN/service_packages/CCS/bin/ccsCDRTrimFilesStartup.sh
3 Database EDR records
  /IN/service_packages/CCS/bin/cdrDeletionStartup.sh

smsCdrProcess.sh

ACS EDR archiving is handled by the **smsCdrProcess.sh** script which is launched from **crontab** once per day (usually midnight GMT).

It contains a number of configurable parameters, but generally it will simply call the **smsProcessCdr** binary from **/IN/service_packages/SMS/bin**, move CDRs from **SMS/cdr/received** to **SMS/cdr/processed**, and delete files older than 31 days.

The script can be altered to perform some additional tasks, e.g. placing a suffix on the final filenames, but in general it is quite rudimentary.

ccsCDRTrimFilesStartup.sh

Billing EDR archiving is handled by the **ccsCDRTrimFilesStartup.sh** script which is launched from **crontab** once per day (usually midnight GMT).

The script calls the ccsCDRTrimFiles binary, which will remove EDR files from the specified directory that are older than the specified amount of days.

In general, this is **/IN/service_packages/CCS/logs/CDR-store**, and files older than 30 days.

cdrDeletionStartup.sh

Billing EDRs are also removed from the database after a set period of time.

This is controlled by the **/IN/service_packages/CCS/bin/cdrDeletionStartup.sh** script, which runs a simple cleanup script on the database once per day from **crontab** (usually 03:15 GMT).

The script itself can be found in **/IN/service_packages/CCS/bin/cdrDeletion.sql**.

In general, this will be configured to remove any EDRs older than seven days.

Customer Specific Processing

From time to time, a customer may have some more specific requirements for EDR files, rather than just storing on the NCC SMS for a number of days before deleting.

Reasons for this include:

- ACS EDR reconciliation for billing
- BE EDR transfer to mediation server for processing to third party front-end
- More thorough archival flexibility:
  - Splitting EDR into daily files
  - Compression
  - Longer retention

If a customer specific processor or archiver is in place, this is usually indicated by some of the following:

- smf_oper crontab has smsCdrProcessor entry commented or reading/writing to a different location.
- ccs_oper crontab has ccsCDRLoader reading/writing to a different location.
- New "cdrArchiver.sh options" in the *_oper crontabs and bin directories.
- Modified directory structure in **CCS/logs** or **SMS/cdr**.
- Separate mount point for EDR, for example,. **/global/EDR** or similar.

Provisioning Interface (PI)

Overview

The Provisioning Interface (PI) runs as a service on the SMS.

PI is primarily responsible for handling external requests for data on the NCC platform. It will accept an external connection (supporting a number of protocols), provide authentication and respond to requests for subscriber information.

This includes simple interactions such as balance queries and performing recharges through to creating new subscribers and issuing commands that trigger through to a control plan to perform complex Business Processing Language (BPL) interactions.

Supported protocols

Supported protocols are:

- XML
- SOAP
- Native Oracle Syntax (plain-text)

Processes using PI

There are a number of processes responsible for running the PI. They are as follows:

- PImanager
  Responsible for starting and stopping PIprocess instances, and the PIbeClient.
- PIprocess
  Each listening port is run through a PIprocess
  Handles requests and executes commands
  Each port has a defined protocol it is listening for:
  - XML
  - SOAP
  - Standard.
- PIbeClient
  Interacts with the billing engine.
- smsTrigDaemon
  Used when the PI needs to trigger a control plan on the SLC, usually for a BPL command.

Command delivery

Each set of commands for the PI are delivered in a separate package for each product, for example, ccsPI for CCS and billing commands, ccsACS for ACS commands.

Commands are issued via the required protocol, based on the configuration shown in the SMS screens under **Services > Provisioning > Administraton > Ports** tab. Each defined port handles one protocol type only.

Command structure

The basic structure of a PI command is:

```
Command=Action:Parameter1=Value1,Parameter2=Value2,…;
```

Tip: The semi-colon terminates and executes the command.

Some example *Command=Action* combinations are:

CCSCD1=QRY

Subscriber query

CCSCD1=ADD

Add Subscriber

CCSCD1=CHG

Change Subscriber

CCSCD1=DEL

Delete Subscriber

CCSCD3=RCH

Recharge Account or Voucher

Note: Each action has a specific set of expected parameters. See the relevant PI commands guide for more information.

Command responses

Each request will receive either a positive or negative acknowledgment (ACK or NACK). The response will generally contain further information, although in some cases the response will just be an ACK.

Running a PI session

Use the following process to access the PI.

First, connect to the PI through telnet to a valid port, enter:

```
$ telnet server port
```

…

Tip: Escape character is '**^]**'.

There is no prompt, but the first interaction the PI expects is a username and password, terminated with a semi-colon:

user,*password*;

```
ACK;
```

This indicates the connection is successful, if not the response would be in the negative:

user,*password*;

```
NACK,72-INVALID LOGON - username,password;Connection to localhost closed by foreign
host.
```

Once successfully connected, commands can be executed. See the relevant PI commands guide for more information on available commands.

Some example Native Oracle Syntax commands and responses are shown below:

Subscriber Query

This command runs a subscriber query, enter:

```
CCSCD1=QRY:MSISDN=12345;
```

Response:

```
CCSCD1=QRY:ACK:MSISDN=12345,ACCOUNT_NUMBER=1012345,PRODUCT=POSTPAID,SERVICE_PROVIDER
=ORACLE,STATUS=A,CREATION_DATE=20100721044959,WALLET_EXPIRY_DATE=,BALANCE_EXPIRY_DAT
E=20111106040500,BALANCE_OFFSET_DATE=,BALANCE=500,INITIAL_BALANCE=0,LANGUAGE=english
,LAST_RECHARGE_DATE=20101115160315,LAST_CC_RECHARGE_DATE=,LAST_USE_DATE=201011220440
14,LAST_RECHARGE_AMOUNT=0,PREV_WALLET_EXPIRY_DATE=20100701022600,PREV_BALANCE_EXPIRY
_DATE=20111106040500,PREV_BALANCE=8590,LAST_EXP_CREDIT=14950,TOTAL_EXP_CREDIT=26950,
LAST_EXP_DATE=20101106023900,FIRST ACTIVATION
DATE=20100726035409,LAST_STATE_CHANGE_DATE=20101011201347,LAST_STATE_CHANGE_REASON=,
BYPASS_NUMBER=,WALLET_TYPE=Primary,CHARGING_DOMAIN=1,FFD=,FFN=,FDN=,CUG=,CURRENCY=NZ
D,FREE_SWAPS_REMAINING=0,LAST_SWAP_RESET_DATE=;
```

Balance Query - all balances

This command runs an all balance query, enter:

```
CCSCD1=QRY:MSISDN=12345,BALANCE_TYPE=ALL;
```

Response:

```
CCSCD1=QRY:ACK:MSISDN=12345,ACCOUNT_NUMBER=1012345,PRODUCT=POSTPAID,STATUS=A,WALLET_
EXPIRY_DATE=,LANGUAGE=english,BALANCES=General Cash:500:20111106040500|Free
SMS:5:20101122173422;
```

Balance Query - specific balance

This command runs a specific balance query, enter:

CCSCD1=QRY:MSISDN=12345,BALANCE_TYPE=Free SMS,LIST_TYPE=BALANCE|BALANCE_EXPIRY_DATE;

Response:

```
CCSCD1=QRY:ACK:MSISDN=12345,ACCOUNT_NUMBER=1012345,BALANCE_EXPIRY_DATE=2010112217342
2,BALANCE=5;
```

Note: Balance types such as `General Cash`, or `Free SMS` are configurable, so the output may vary from platform to platform.

Voucher Type Recharge

This command runs a voucher type recharge request, enter:

CCSCD3=RCH:RECHARGE_TYPE=VoucherType,REFERENCE=100 Free SMS,MSISDN=12345;

Response:

```
CCSCD3=RCH:ACK;
```

Business Processing Language

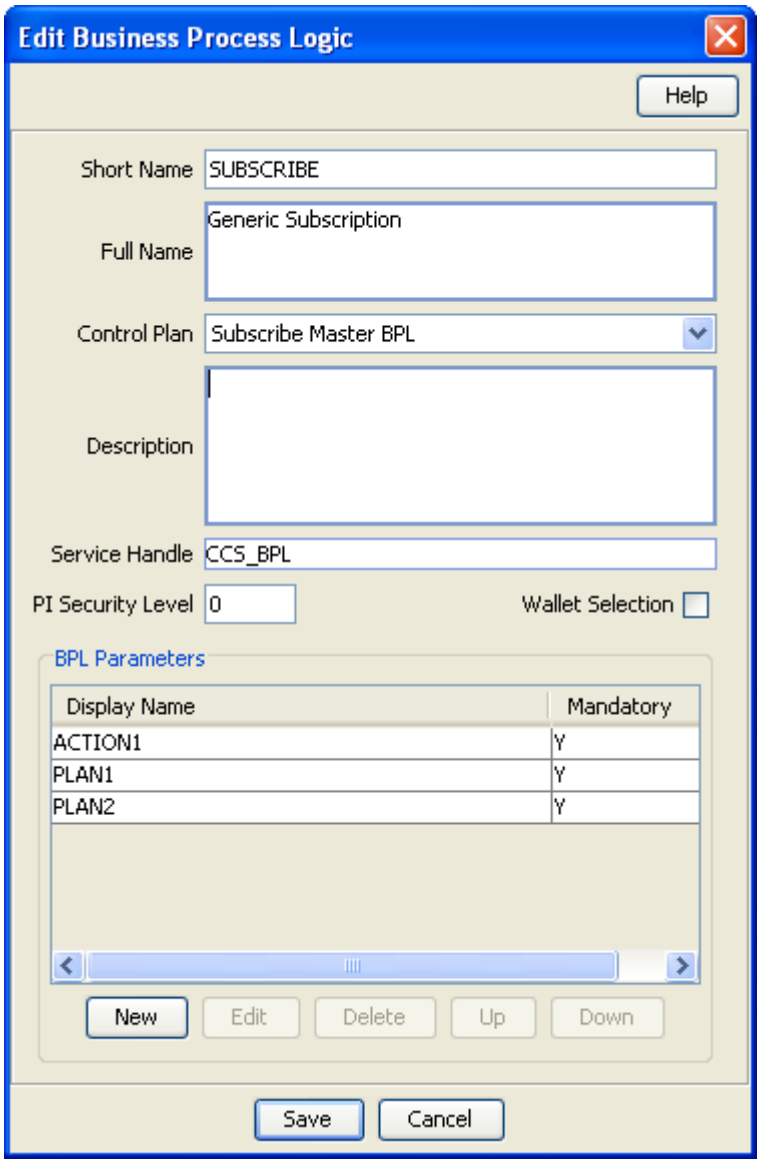
Introduction

More complex and advanced commands can be run by using Business Processing Language (BPL). A BPL command is configured through the SMS UI under **Services > Prepaid Charging > Task Management**.

For full details on configuring BPL, see the *CCS User's Guide*.

Example BPL configuration

In this example Edit Business Process Logic screen, a command name, parameters and control plan are defined:



When the command is run via the PI, it will trigger the specified Service and Control Plan, which will handle the complicated interaction required, and set a response code using a Cause Value inside a Disconnect node in the associated Control Plan. Calling BPL is done using the command CCSBPL=EXE. For example:

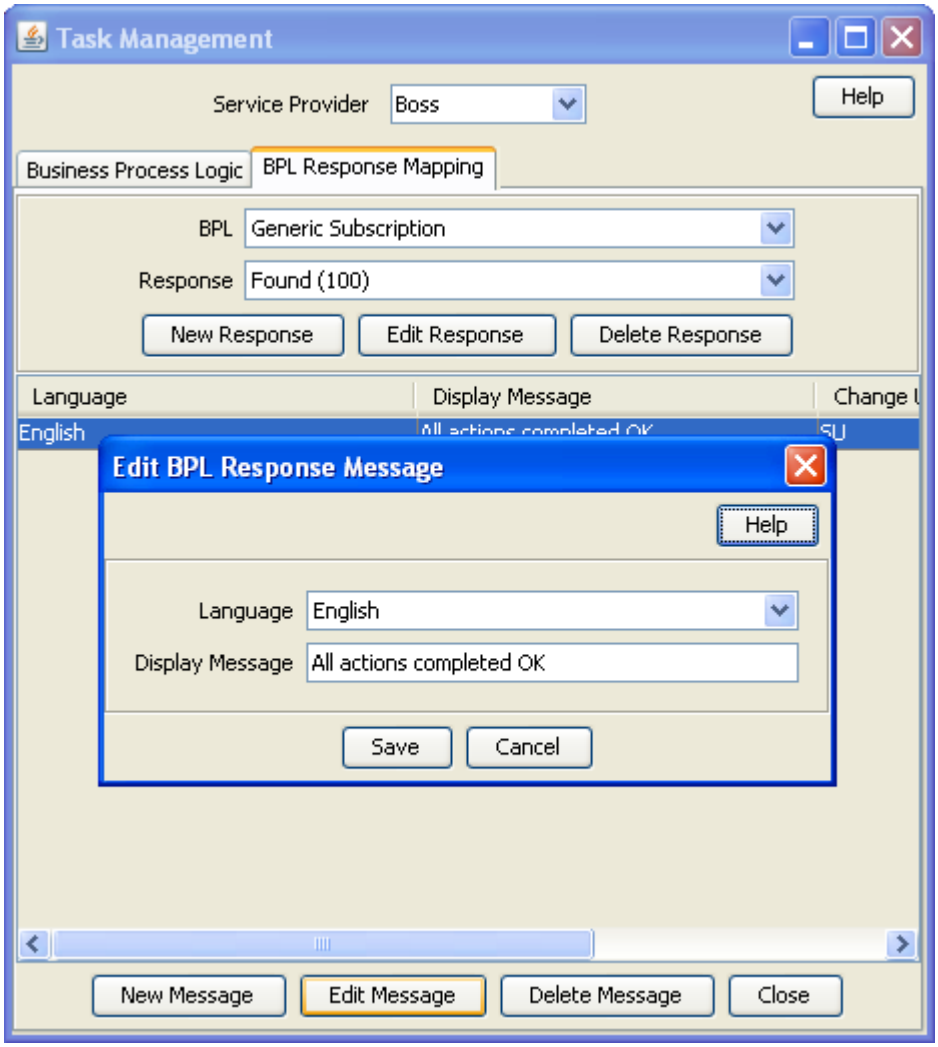
```
CCSBPL=EXE:MSISDN=12345,BPL=SUBSCRIBE,EXT1=0,EXT2=1,EXT3=0;

 CCSBPL=EXE:ACK:302-All actions completed OK;
```

Note: The extension parameters are defined in the BPL record and correspond to the extension parameters (4 to 8) defined in the acs.conf configuration file.

Example Disconnect Cause mappings

Mappings between Disconnect Cause and the message returned to the PI is also done through **Services > Prepaid Charging > Task Management**.

302 is the PI result code for a successful BPL command.



So in this example, the Control Plan exited with a Disconnect Cause of 100, which was mapped to the display message of "All actions completed OK".

Chapter 6

Voucher and Wallet Server (VWS)

Overview

Introduction

This chapter explains how and why the VWS servers are used.

In this chapter

This chapter contains the following topics.

Voucher and Wallet Server Overview

Introduction

Any interactions by subscribers that require payment are processed, at some point, by the VWS.

The VWS database contains important subscriber information, and all information relating to their balances, promotions and vouchers. Like the SLC, all the main processing takes place inside the SLEE; the main processes responsible for handling billing interactions are:

- beVWARS
- beServer
- beSync
- beGroveller

Some NCAs may also be configured to handle other interactions between VWS and SLC (for example, DAP, OSD). Understanding when these interactions are invoked will also aid the troubleshooting process.

Billing pairs

VWSs are configured in billing pairs, rather than the standard N+1 model that the SLCs follow. One machine acts as the primary, the other secondary, whereby the primary processes all traffic, and syncs the information to the secondary.

In the event of an outage of the primary, traffic is handed to the secondary for processing. When the primary is brought back into operation, it requests all the updates it is missing from the secondary and begins to resync. Only once the syncing process is complete, will the primary resume operation and begin to handle all the traffic again.

From a subscriber's perspective this is a completely transparent process. If an initial request is handled by the primary and it is taken out of service by the time a subsequent reservation comes in, the secondary will step in and handle the reservation. Regardless of the status of the primary, the secondary will continue to handle billing for any calls that "bounce" to it until the calls are complete.

Processes location

Billing processes are located in **/IN/service_packages/E2BE/bin**. Each process writes to its own log file in **E2BE/tmp**. Each instance of beVWARS also writes to its own log file.

General Billing Terms

Explanation of a few basic billing terms will aid understanding of how the NCC VWS works, and the information provided in the rest of the VWS section.

- Subscriber
  The end user
- Wallet
  A container for a subscriber's buckets.
  Many-to-many relationship with a subscriber:
  - A subscriber can have one primary and one secondary wallet
  - A wallet may be shared between many subscribers
- Bucket
  A record of a balance for a subscriber inside a wallet.
  One-to-many relationship with a wallet:
  - A wallet can have many buckets
  - A bucket can only belong to one wallet
- Periodic Charge
  Stored in a regular bucket, but with additional references to the periodic charge.
  Uses **Expiry** as the renewal date, that is, when the bucket expires the periodic charge will renew.
  Uses **Value** to store the current periodic charge state:
  - 1 - Terminated
  - 2 - Unsubscribed
  - 3,4,5,6 - Active (varying states of active)
  - 7,8,9 - Grace (varying states of Grace Period)
- ACK/NACK
  Standard nomenclature for a successful (ACK) or failed (NACK) response from the VWS.

beServer process

The beServer acts as a central contact point for connecting clients to the billing engine. Essentially any interaction with beVWARS will first go through beServer.

beServer maintains a list of currently connected clients and handles new connections.

Client list

Different clients include:

- beClient (SLC)
- PIbeClient (PI on the SMS)
- ccsBeOrb (Java screens on the SMS)
- beGroveller (background process for keeping un-used wallets up to date)

Listening port

The VWS server listens on port 1500 for incoming connections, and uses the Oracle Escher protocol for communications. If troubleshooting is required for billing traffic - port 1500 will contain the conversation that happens on the wire between NCC components.

Client ID

Each connecting client has its own unique client ID. This is derived from a hash of the client name, specified by the configuration of the incoming client process.

Upon connection, beServer logs the client name and client id, which can be a useful reference when trying to determine the client for an EDR, for example:

```
Nov 23 04:36:11 ube01 beServer: [ID 839465 user.notice] beServer(21708) NOTICE: Client
'slc01_ccsBeClient' (ClientId 87783972) has connected
Nov 23 04:36:13 ube01 beServer: [ID 839465 user.notice] beServer(21708) NOTICE: Client
'PIbeClient' (ClientId 161986004) has connected
```

beVWARS process

beVWARS is the main process handling the work-load on a VWS. It is responsible for all interactions between the subscriber and their funds.

At a rudimentary level, it holds the cache that represents the most up-to-date information about a subscriber balance information, including uncommitted funds (that is, reservations).

Upon a request for a subscriber's wallet, beVWARS will load the subscriber information from the database into cache, and periodically flush and write to the database based on configuration (beVWARS.walletCache{} section in **eserv.config**).

Handlers

beVWARS is reasonably flexible, and will operate using a configured set of Message and Event Handlers.

Message Handlers define how beVWARS will handle message requests from clients (for example, how to handle a voucher recharge request).

Event Handlers (known as plugins) contain a set of instructions to be run on wallets each time an event is triggered (for example, instructions to delete an expired balance).

Plugins example list

Plugins will be run prior to the handlers, so that any maintenance has been run prior to call connection. Plugins include:

- beVWARSExpiry.so
  Processes expired buckets, ensuring that expired funds are removed from the database
- beVWARSMergeBuckets.so
  Manages the number of buckets a wallet is allowed. If the maximum is hit, the new bucket will be merged into an existing one instead
- ccsVWARSExpiry.so
  Maintains CCS Wallet States, for example.
  - Moves Dormant Wallets to Active when they are used
  - Deletes Terminates Wallets after a configurable period of time
- ccsVWARSActivation.so
  Activates Wallets including initial credits
- ccsVWARSPeriodicCharge.so
  Handles all PeriodicCharge interactions and state changes
- ccsNotification.so
  Creates real-time notifications

Handlers example list

Handlers include:

- ccsVWARSReservationHandler.so
  Performs the UBE-side processing of all messages relating to chargeable call processing including calculating tariffs
- ccsVWARSNamedEventHandler.so
  Performs the UBE-side processing of messages relating to named events. This includes:
  - Returning the cost for an event class and event name combination
  - Generating named event EDRs.
- ccsVWARSRechargeHandler.so
  Handles General Wallet Recharges
- beVWARSCCDRHandler.so
  Handles EDR generation in some situations where one would not usually be generated (can be specifically requested by a BE Client)
- ccsVWARSWalletHandler.so
  Performs the UBE side processing of all messages relating directly to wallets. This includes:
  - Wallet Information (WI) - responds with wallet information
  - Wallet Create (WC) - creates new wallets
  - Wallet Update (WU) - updates wallets
  - Wallet Delete (WD) - deletes existing wallets and corresponding buckets
  - Bad PIN updates (BPIN) - updates Bad PIN balance if the wallet has one.

  EDRs are produced for all Wallet updates (create/modify/delete/recharge) with the details of the change
- ccsVWARSVoucherHandler.so
  Performs the Billing Engine side processing of messages directly relating to vouchers.
  This includes voucher reservation/commit, alteration and deletion

beVWARS scalability

beVWARS is a scalable process, and runs multiple instances on the NCC platform.

As the beVWARS contains the most up to date information about a Wallet, the beServer needs to ensure that not only is the workload even, but subsequent requests for wallet information must always go to the same beVWARS instance.

Workload spreading

The algorithm for this is *WalletID* MOD *Total_beVWARS_Instances*.

The Wallet ID is essentially an identity field in the database, and will increment in a way that ensures even workload.

The number of instances created is determined by **/IN/service_packages/SLEE/etc/SLEE.cfg**. For example:

```
INTERFACE=beVWARS0 beVWARS.sh /IN/service_packages/E2BE/bin EVENT
INTERFACE=beVWARS1 beVWARS.sh /IN/service_packages/E2BE/bin EVENT
INTERFACE=beVWARS2 beVWARS.sh /IN/service_packages/E2BE/bin EVENT
INTERFACE=beVWARS3 beVWARS.sh /IN/service_packages/E2BE/bin EVENT
INTERFACE=beVWARS4 beVWARS.sh /IN/service_packages/E2BE/bin EVENT
INTERFACE=beVWARS5 beVWARS.sh /IN/service_packages/E2BE/bin EVENT
INTERFACE=beVWARS6 beVWARS.sh /IN/service_packages/E2BE/bin EVENT
INTERFACE=beVWARS7 beVWARS.sh /IN/service_packages/E2BE/bin EVENT
INTERFACE=beVWARS8 beVWARS.sh /IN/service_packages/E2BE/bin EVENT
INTERFACE=beVWARS9 beVWARS.sh /IN/service_packages/E2BE/bin EVENT
INTERFACE=beVWARS10 beVWARS.sh /IN/service_packages/E2BE/bin EVENT
INTERFACE=beVWARS11 beVWARS.sh /IN/service_packages/E2BE/bin EVENT
INTERFACE=beVWARS12 beVWARS.sh /IN/service_packages/E2BE/bin EVENT
INTERFACE=beVWARS13 beVWARS.sh /IN/service_packages/E2BE/bin EVENT
```

```
INTERFACE=beVWARS14 beVWARS.sh /IN/service_packages/E2BE/bin EVENT
INTERFACE=beVWARS15 beVWARS.sh /IN/service_packages/E2BE/bin EVENT
```

However, it is also defined in the BE{} section of **eserv.config**, which saves time and complexity:

```
numVWARS = 16
```

beSync process

beSync is responsible for keeping the Primary and Secondary Billing Engine pair in the same state at any given time.

Each time the beVWARS performs an interaction on the database, a sync file is written to the beSync repository - usually **/IN/service_packages/E2BE/sync/<beVWARS Instance #>**.

These sync files are processed by beSync and used to write the information to the other Billing Engine in the pair. After they are completed the files are deleted. The number of files waiting to be processed can be indicative of an unsynchronized Billing Engine pair.

When only one node in a Billing Engine pair is running, all the information yet to be sent to the other node will start collecting in the beSync repository and will continue to do so until it comes back up.

beSync will also transmit information about new reservations, ensuring that both Billing Engine nodes are aware of any ongoing uncharged funds. The result is that the Primary can go down mid-call, and the subscriber will not be aware that anything has gone wrong.

Billing Engine startup

Upon startup, beSync will ask each local beVWARS for their last written sequence number.

beSync uses this, and the BE_VWARS_SEQ_NUM table in the database, to track what updates are yet to be synced between the machines in the pair and begin syncing immediately.

Since beServer will not accept any connections until the Billing Engine is completely up to date, it can be prudent to monitor the backlog in **E2BE/sync** to see how it is progressing.

beSync will also collect reservations from the other node, so it is completely up to date.

beGroveller process

beGroveller is responsible for searching the database for unused wallets, and sending them to the correct beVWARS process when requested.

During normal processing, events are triggered only when a subscriber interacts with the wallet. Some events (such as expiry and periodic charges) should be triggered regardless of whether the wallet has been used by a subscriber recently or not.

Event processing

In order to process these events, beGroveller collects and sends lists of wallets IDs to beVWARS for processing. This processing triggers any events which are due to occur in the same way a normal interaction would, except wallets triggered from beGroveller lists do not trigger any message handlers.

No-processing times

In general it is not imperative that buckets are expired in real-time, and grovelling unused wallets consumes resources that beVWARS would otherwise be using to process regular traffic.

For this reason beGroveller contains some configuration (beGroveller{} section in **eserv.config**) for running only during certain times of the day:

```
noProcessingTimes = [
    { startsAt = "06:00", endsAt = "09:30" }
]
```

During these no-processing times when the beVWARS asks for more wallets to grovel, beGroveller will report that there are none.

beGroveller scalability

Like beVWARS, beGroveller is also a scalable process, and runs multiple instances on the NCC platform. Although the beGroveller uses the same algorithm for calculating which beGroveller is going to serve a particular wallet, it does not need to run the same number of instances as beVWARS - it will often run much less.

Both beVWARS and beGroveller are able to determine the instances of the other, and will access the appropriate instance accordingly.

Workload spreading

The algorithm for this is *WalletID* MOD *Total_beGroveller_Instances*

The number of instances created is determined by **/IN/service_packages/SLEE/etc/SLEE.cfg**. For example:

```
INTERFACE=beGroveller0 beGroveller.sh /IN/service_packages/E2BE/bin EVENT
INTERFACE=beGroveller1 beGroveller.sh /IN/service_packages/E2BE/bin EVENT
INTERFACE=beGroveller2 beGroveller.sh /IN/service_packages/E2BE/bin EVENT
INTERFACE=beGroveller3 beGroveller.sh /IN/service_packages/E2BE/bin EVENT
```

However, it is also defined in the BE{} section of **eserv.config** - which saves time and complexity:

```
numGrovellers = 4
```

Useful Commands and Scripts

Introduction

This topic lists a few useful scripts for investigating a billing issue.

Without access to the SMS screens (quite common when dealing with a production environment) it can be difficult to ascertain basic information about a subscriber and their wallets.

showCli.sh

This script displays the basic information about a CLI in the database. At a glance it will tell you:

- whether the number exists in the database, and
- if so, some useful information about their entry (including defined wallet IDs)

It can also be used to compare the information on the SMS database with the VWS (or SLC) database; thus allowing a quick check that replication is functioning.

All databases should always have the same information. If there are differences, there could be an issue with replication, and a full resync may be required.

Machine Type: SMS, VWS, SLC

Location: **/IN/service_packages/CCS/bin/showCli.sh**

Usage: `./showCli.sh` *`CLI/MSISDN`*

Example Output:

```
/IN/service_packages/CCS/bin$ ./showCli.sh 12345
CLI        ACCT_REFERENCE NAME                           WALLET ACCOUNT_NUMBER
---------- -------------- ------------------------------ ------ --------------
12345              2143                        Primary   2143           1656
```

showWallet.sh

This script displays the contents of a subscriber's wallet. At a glance it will give you information on the wallet state, and what buckets are contained within that wallet, including expiry dates and references.

It can also be used to compare the information on the Primary and Secondary Billing Engine databases - thus allowing a quick check that the Billing Engine pair is in sync. Unless one of the nodes has been down for an extended period of time, and is still being brought into sync (in which case it will not be handling traffic), the results on both Billing Engine nodes should have the same information in the database.

Note: beVWARS operates a cache, so the database will not necessarily reflect a subscriber's available funds. Ongoing reservations are not written to the database until they are confirmed. A subscriber could legitimately appear to have sufficient funds for a call in the database, but have insufficient funds due to an in-progress reservation.

Machine Type: VWS

Location: **/IN/service_packages/E2BE/bin/showWallet.sh**

Usage: `./showWallet.sh` *`WalletID`*

Example Output:

```
/IN/service_packages/E2BE/bin$ ./showWallet.sh 2143
Showing wallet 2143 on e2be_admin

        ID MAX_CONCURRENT S NEVER_EXPIRES EXPIRY         NEVER_ACTIVATED ACTIVATION_DAT
---------- -------------- - ------------- -------------- --------------- --------------
      2143             10 A             1 20100701022600               0 20100726035409


    WALLET BALANCE_TYPE L MINIMUM_CREDIT
---------- ------------ - --------------
      2143           22 D              0
      2143           27 D              0

2 rows selected.


        ID     WALLET BALANCE_TYPE NEVER_EXPIRES EXPIRY              VALUE NEVER_USED LAST_USE
REFERENCE                                                START_DATE
---------- ---------- ------------ ------------- -------------- ---------- ---------- ----------
---- ---------------------------------------------------------- --------------
     16734       2143           22             0 20111106040500        500          0
20101122043401                                                    19700101000000
     16998       2143           27             0 20101223033753        100          0
20101123033753
     17078       2143           27             0 20101223041742        600          0
20101123041742

6 rows selected.
```

Tip: The output is wrapped, and generally easier to read within a terminal window

Chapter 7

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

To troubleshoot the product, first you must identify the system which is responsible for the service that needs troubleshooting.

As explained in the Product System Architecture section, there are three main server components in the NCC:

- Service Logic Controller (SLC)
  The SLC is responsible for most real-time service processing (for example, voice/SMS/data). Call handling issues are likely to require troubleshooting on the SLC
- Service Management System (SMS)
  The SMS is responsible for provisioning, data warehousing and replication. Issues specific to certain subscribers, coinciding with important changes to rating, concerning EDRs or with external provisioning (via the Provisioning Interface (PI)) require troubleshooting on the SMS.
- Voucher and Wallet Server (VWS)
  The VWS is responsible for voucher redemption and call rating (this includes balance management and promotions tracking). Issues concerning subscribers' balances, top-ups and vouchers are likely to require troubleshooting on the VWS.

Important notice

Please note that NCC packages are complete versions and were tested as such.

If you have any questions or problems, please contact Oracle.

General tools

The following information is not specific to any particular type of node, and can be helpful when investigating any problem situation.

The list of processes is built from inittab, and will highlight any defined that are not running. If a SLEE is present, its configuration will be parsed, and SLEE processes included in the list.

Process status

There are a few basic checks that can be run on any of the machines, which are provided as part of the supportScp (SLC/VWS) or supportSms (SMS) packages. These give you a quick look at what processes are running.

Example - pslist

This example shows the pslist command used with no parameters.

Command:

```
$ pslist
```

Result:

```
------------------------ Thu Oct 24 04:56:53 GMT 2010 --------------------------
C APP  USER       PID PPID    STIME COMMAND
1 ACS  acs_oper  1004    1   04-Oct N/service_packages/ACS/bin/acsCompilerDaemon
1 ACS  acs_oper  1008    1   04-Oct /service_packages/ACS/bin/acsProfileCompiler
1 ACS  acs_oper 13833    1 00:12:38 ice_packages/ACS/bin/acsStatisticsDBInserter
1 OSD  acs_oper  1047    1   04-Oct /service_packages/OSD/bin/osdWsdlRegenerator
1 CCS  ccs_oper  1011    1   04-Oct /IN/service_packages/CCS/bin/ccsCDRLoader
1 CCS  ccs_oper  1033    1   04-Oct service_packages/CCS/bin/ccsCDRFileGenerator
1 CCS  ccs_oper 11411    1   13-Oct /IN/service_packages/CCS/bin/ccsBeOrb
2 CCS  ccs_oper  1406 1043   04-Oct IN/service_packages/CCS/bin/ccsProfileDaemon
1 CCS  ccs_oper  9413    1   04-Oct /IN/service_packages/CCS/bin/ccsChangeDaemon
1 EFM  smf_oper   995    1   04-Oct /IN/service_packages/EFM/bin/smsAlarmManager
1 PI   smf_oper  1080    1   04-Oct /IN/service_packages/PI/bin/PImanager
6 PI   smf_oper  1319 1080   04-Oct PIprocess
1 PI   smf_oper  9186 1080   04-Oct PIbeClient
2 SMS  smf_oper  6173    1   21-Oct /IN/service_packages/SMS/bin/smsMaster
1 SMS  smf_oper   941    1   04-Oct /IN/service_packages/SMS/bin/smsAlarmRelay
1 SMS  smf_oper   943    1   04-Oct /IN/service_packages/SMS/bin/smsNamingServer
1 SMS  smf_oper   944    1   04-Oct IN/service_packages/SMS/bin/smsReportsDaemon
1 SMS  smf_oper   946    1   04-Oct /service_packages/SMS/bin/smsReportScheduler
1 SMS  smf_oper   947    1   04-Oct /IN/service_packages/SMS/bin/smsAlarmDaemon
1 SMS  smf_oper   948    1   04-Oct N/service_packages/SMS/bin/smsStatsThreshold
1 SMS  smf_oper   949    1   04-Oct /IN/service_packages/SMS/bin/smsTaskAgent
1 SMS  smf_oper   969    1   04-Oct /IN/service_packages/SMS/bin/smsTrigDaemon
2 SMS  smf_oper   979    1   04-Oct /IN/service_packages/SMS/bin/smsConfigDaemon
1 SMS  smf_oper   980    1   04-Oct N/service_packages/SMS/bin/smsStatsDaemonRep
total processes found = 32 [ 32 expected ]
================================= run-level 3 ==================================
```

Example - pslist -d

This example shows the pslist command used with the **-d** parameter. From time to time, processes will be added to or removed from inittab/SLEE. The -d parameter instructs pslist to reconstruct the list.

Command:

```
$ pslist -d
```

Result:

```
Scanning input file.
   [ /etc/inittab ]
Scanning input file.
   [ /IN/service_packages/SLEE/etc/SLEE.cfg ]
Info: Did not find SLEE config file [ /IN/service_packages/SLEE/etc/SLEE.cfg ]
      Does the SLEE application exist on this machine?

<----
###############################################################################
# pslist: default process list configuration (plc) file used to match and  #
# display running processes.                                                  #
```

```
# File creation time: Thu Nov 13 04:19:29 GMT 2008                      #
# Lines beginning with a hash (#) character are ignored.                #
# $1="grouped-apps name (max 5-char)" $2="regex of process" [$3+=comments] #
#########################################################################
ACS   acs_oper.*\/IN\/service_packages\/ACS\/bin\/acsCompilerDaemon            inittab
ACS   acs_oper.*\/IN\/service_packages\/ACS\/bin\/acsProfileCompiler           inittab
ACS   acs_oper.*\/IN\/service_packages\/ACS\/bin\/acsStatisticsDBInserter      inittab
CCS   ccs_oper.*\/IN\/service_packages\/CCS\/bin\/ccsBeOrb                     inittab
CCS   ccs_oper.*\/IN\/service_packages\/CCS\/bin\/ccsCDRFileGenerator          inittab
CCS   ccs_oper.*\/IN\/service_packages\/CCS\/bin\/ccsCDRLoader                 inittab
CCS   ccs_oper.*\/IN\/service_packages\/CCS\/bin\/ccsChangeDaemon              inittab
CCS   ccs_oper.*\/IN\/service_packages\/CCS\/bin\/ccsProfileDaemon             inittab
EFM   smf_oper.*\/IN\/service_packages\/EFM\/bin\/smsAlarmManager              inittab
OSD   acs_oper.*\/IN\/service_packages\/OSD\/bin\/osdWsdlRegenerator           inittab
PI    smf_oper.*PIbeClient                                                    inittab: PI
Manager child process
PI    smf_oper.*PIprocess                                                     inittab: PI
Manager child process
PI    smf_oper.*\/IN\/service_packages\/PI\/bin\/PImanager                    inittab
SMS   smf_oper.*\/IN\/service_packages\/SMS\/bin\/smsAlarmDaemon              inittab
SMS   smf_oper.*\/IN\/service_packages\/SMS\/bin\/smsConfigDaemon             inittab
SMS   smf_oper.*\/IN\/service_packages\/SMS\/bin\/smsMaster                   inittab
SMS   smf_oper.*\/IN\/service_packages\/SMS\/bin\/smsNamingServer             inittab
SMS   smf_oper.*\/IN\/service_packages\/SMS\/bin\/smsReportScheduler          inittab
SMS   smf_oper.*\/IN\/service_packages\/SMS\/bin\/smsReportsDaemon            inittab
SMS   smf_oper.*\/IN\/service_packages\/SMS\/bin\/smsStatsDaemonRep           inittab
SMS   smf_oper.*\/IN\/service_packages\/SMS\/bin\/smsStatsThreshold           inittab
SMS   smf_oper.*\/IN\/service_packages\/SMS\/bin\/smsTaskAgent                inittab
SMS   smf_oper.*\/IN\/service_packages\/SMS\/bin\/smsTrigDaemon               inittab
---->
Default process list configuration file created.
 [ /IN/service_packages/SMS/tmp/ps_processes.testusms.plc ]
```

Process configuration

Configuration for NCC products and processes are made almost exclusively in the file **/IN/service_packages/eserv.config**.

The file is broken down into sections and subsections, grouped together by {} brackets. Each product comes with an example **eserv.config** inside their respective <Product>/etc directories, and each configuration option is documented in the associated Technical Guide.

There are some exceptions, notably ACS and SLEE, which have some separate configuration files in **/IN/service_packages/ACS/etc/acs.conf** and **/IN/service_packages/SLEE/etc/SLEE.cfg** respectively.

Some NCA interface configuration is also housed in a separate file; for example, for SIGTRAN interfaces (sua_if/m3ua_if) the configuration is often specified in **/IN/service_packages/SLEE/etc/sigtran.config** or *interface*_**service.config**.

Note: Processes also have command line arguments, which are passed in the calling shell script - normally named **/IN/service_packages/<Product>/bin/***ProcessName***Startup.sh**.

Remote Diagnostic Agent

Introduction

Remote Diagnostic Agent (RDA) is an Oracle cross-product diagnostic tool used to help Oracle engineers in troubleshooting and analyzing issues.

RDA supports Oracle Communications Network Charging and Control (NCC).

For a more general usage guide of the Remote Diagnostic Agent tool, please refer to the references included in the following sections.

Installing RDA

To install RDA, please review My Oracle Support Note **314422.1**.

For consistency across all platforms and NCC nodes, upload the RDA package to the **/IN/service_packages/SUPPORT/** directory and proceed with the installation from this location.

To install RDA on your NCC nodes:

Step	Action
1	Navigate to the directory where you downloaded the RDA package. For example, **/IN/service_packages/SUPPORT/**
2	Uncompress the RDA file as the *smf_oper* user. This will create a subfolder named **rda** in the current folder, containing all files for RDA execution.
	Note: Due to restrictive security policies, RDA should not be installed/run as the root user - *smf_oper* should have all accesses and permissions it needs.

Configuring RDA

To set up the RDA profile and activate the NCC module, navigate to the RDA directory and use the following command:

```
smf_oper@server$./rda.sh -vdSp Com_NCC
```

The tool will prompt you with a few questions regarding your environment. Most default answers should be sufficient for your environment. However, you must select the **Yes** option to collect information from your Oracle database. Review each prompt ensuring that the responses are specific for your environment.

Additionally:

- The prompt about ADDM, AWD, and ASH is necessary due to restricted use of these features for licensing reasons.
- The *system* user should not connect as *sysdba* if the *smf_oper* user in your environment does not have sysdba permissions for your NCC database.
- Configuration, except passwords, is stored by the tool for future execution of the RDA tool.

Example RDA Output

Here is an example RDA output:

```
bash-4.1$ ./rda.sh -vdSp Com_NCC
Setting up ...
--------------------------------------------------------------------------------
S000INI: Initializes the Data Collection
--------------------------------------------------------------------------------
RDA uses the output file prefix to identify all files belonging to the same
data collection. The prefix must start with a letter and must contain only
alphanumeric characters.

Enter the prefix to be used for all the generated files
Hit 'Return' to accept the default (RDA)
>

Enter the directory used for all the files to be generated
Hit 'Return' to accept the default (/IN/service_packages/SUPPORT/rda/output)
>

Do you want to keep report packages from previous runs (Y/N)?
Hit 'Return' to accept the default (N)
>

Enter the Oracle home to be used for data analysis
Hit 'Return' to accept the default (/u01/app/oracle/product/12.1.0)
>

Enter the network domain name for this server
Hit 'Return' to accept the default (us.oracle.com)
>

--------------------------------------------------------------------------------
S010CFG: Collects Key Configuration Information
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
S090OCM: Set up the Configuration Manager Interface
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
S909RDSP: Produces the Remote Data Collection Reports
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
S919LOAD: Produces the External Collection Reports
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
S999END: Finalizes the Data Collection
--------------------------------------------------------------------------------
```

```
------------------------------------------------------------------------------
---
S100OS: Collects the Operating System Information
------------------------------------------------------------------------------
---
------------------------------------------------------------------------------
---
S105PROF: Collects the User Profile
------------------------------------------------------------------------------
---
------------------------------------------------------------------------------
---
S110PERF: Collects Performance Information
------------------------------------------------------------------------------
---
Can ADDM, AWR, and ASH be used (Y/N)?
Hit 'Return' to accept the default (Y)
>

------------------------------------------------------------------------------
---
S120NET: Collects Network Information
------------------------------------------------------------------------------
---
Do you want RDA to perform the network ping tests (Y/N)?
Hit 'Return' to accept the default (N)
>

------------------------------------------------------------------------------
---
S122ONET: Collects Oracle Net Information
------------------------------------------------------------------------------
---
------------------------------------------------------------------------------
---
S200DB: Controls Oracle RDBMS Data Collection
------------------------------------------------------------------------------
---
Is the database associated to the current Oracle home (Y/N)?
Hit 'Return' to accept the default (Y)
>

Enter the Oracle SID to be analyzed
Hit 'Return' to accept the default (SMF)
>

Is the INIT.ORA for the database to be analyzed located on this system?
(Y/N)
Hit 'Return' to accept the default (Y)
>

Enter the location of the spfile or the INIT.ORA (including the directory
and
file name)
Hit 'Return' to accept the default
(/u01/app/oracle/product/12.1.0/dbs/initSMF.ora)
>
```

```
Enter an Oracle User ID (userid only) to view DBA_ and V$ tables. If RDA
will
be run under the Oracle software owner's ID, enter a forward slash (/) here,
and enter Y at the SYSDBA prompt to avoid a prompt for the database password
at runtime.
Hit 'Return' to accept the default (system)
   >

   Is 'system' a SYSDBA user (will connect as SYSDBA) (Y/N)?
   Hit 'Return' to accept the default (N)
   >

   ------------------------------------------------------------------------
------
   S201DBA: Collects Oracle RDBMS Information
   ------------------------------------------------------------------------
------
   ------------------------------------------------------------------------
------
   S204LOG: Collects Oracle Database Trace and Log Files
   ------------------------------------------------------------------------
------
   ------------------------------------------------------------------------
------
   S491NCC: Collects Network Charging and Control Information
   ------------------------------------------------------------------------
------
   Enter the full path of the Network Charging and Control home directory
   Hit 'Return' to accept the default (/IN/service_packages)
   >

   WARNING: RDBMS information is collected from Oracle Database only.

   Do you want to collect application information from an Oracle Database
(Y/N)?
   Hit 'Return' to accept the default (N)
   > Y

   Enter the Oracle SID of the database
   Hit 'Return' to accept the default (SMF)
   >
```

```
   Enter an Oracle User ID (userid only) to view application specific
database
   information
   Hit 'Return' to accept the default (smf)
   > smf

   ------------------------------------------------------------------------
------
   S990FLTR: Controls Report Content Filtering
   ------------------------------------------------------------------------
------
           Updating the setup file ...
```

Collecting Data

Use the following recommended flags (or adapt them according to your needs):

`smf_oper@`*`server`*`$./rda.sh -vfCRP`

where *server* is the NCC node where RDA runs and the flags are defined as follows:

- **v**: verbose
- **C**: collect
- **R**: render into html
- **P**: Package contents of output directory into archive
- **f** : force execution of all commands

The script will request the system password and may request a user with the statspack tool installed based on your selections during the configuration.

RDA generates multiple files in the **output/** folder. A zip archive file containing all of the output files is also generated. Download only the zip file from the server where the RDA report was run from the **output/** folder for submission. The script may take several minutes to complete.

Note: Subsequent RDA script execution overwrites the previous reports.

Using Output Immediately

After the archive file is uploaded to Oracle Support, post-processing of the data occurs. The post-processing does not add, remove nor modify the data, it only organizes and applies some formatting. Oracle recommends uploading RDA output files for post-processing. However, it is possible to unzip the file on any computer and directly browse the files.

To optionally view the RDA output immediately before sending the data to Oracle Support, completely unzip the archive and double click on the file named **RDA__start.htm.** This will open the RDA web interface in your default web browser.

Attaching the ZIP Archive to a Service Request

Upload the generated zip file a previously opened Service Request in My Support.

cmnPushFiles/cmnReceiveFiles

cmnPushFiles is responsible for monitoring a location on the SLC/VWS for new files, and will "push" the files to the SMS.

cmnPushFiles is called from inittab, and will run in run-level 3 and generally runs multiple instances.

Each instance will monitor the EDRs of a certain product or process (for example, MM EDRs created by xmsTrigger, ACS EDRs created by slee_acs), however it can also be used to push expiry messages or notifications between machines.

In order for cmnPushFiles to successfully "push" files to the SMS, the network service cmnReceiveFiles must be configured on the SMS in **/etc/inetd.conf and /etc/services**

cmnPushFiles is crucial to the EDR processing chain, and if it is not running or configured incorrectly, then files will build up on the SLC/VWS indefinitely until the system runs out of disk space.

Example - PushFiles

Consider this sample output from a VWS:

```
$ ps -ef | grep Push
```

```
ebe_oper 12479 … cmnPushFiles -d /IN/service_packages/E2BE/logs/CDR-out -r /IN/service_packages/
ccs_oper 12519 … cmnPushFiles -d /IN/service_packages/CCS/logs/expiryMessage/ -r /IN/service_pac
ccs_oper 12480 … cmnPushFiles -d /IN/service_packages/CCS/logs/wallet -r /IN/service_packages/CC
ccs_oper 12482 … cmnPushFiles -d /IN/service_packages/CCS/logs/ccsNotificationWrite/ -r /IN/serv
```

The command response shows there are four instances of cmnPushFiles running.

Using the arguments given to the process, what the process is responsible for can usually be determined:

```
$ pargs 12479
```

```
12479:  cmnPushFiles -d /IN/service_packages/E2BE/logs/CDR-out -r /IN/service_packages/
argv[0]: cmnPushFiles
argv[1]: -d
argv[2]: /IN/service_packages/E2BE/logs/CDR-out
argv[3]: -r
argv[4]: /IN/service_packages/CCS/logs/CDR-in
argv[5]: -h
argv[6]: usms.CdrPush
argv[7]: -F
```

Here we see this cmnPushFiles is taking completed EDRs from **CDR-out** on the VWS and sending them to **CDR-in** on the SMS.

Space issues

If the cmnPushFiles log file (**/IN/service_packages/E2BE/tmp/cmnPushFiles**), or the syslog is reporting insufficient space, checking available space in CDR-out on the VWS and CDR-in on the SMS will be the first step to diagnosing the problem.

Core files

When monitoring a platform, or investigating issues, it is important to check for core files.

Processes running from inittab will be automatically restarted by Solaris, and processes running inside the SLEE will be restarted by the watchdog if they stop running.

If a process cores due to a recurring traffic scenario, it will be restarted and continue to core until the mount point runs out of disk space.

Core file location

The location of core files differs depending on configuration, and how the process was started.

The first thing to check is the output of coreadm, which specifies how the operating system will handle core files.

Multiple core locations

In this example, core files will write to the directory they were called from (in the case of SLEE processes, this will be **/IN/service_packages/SLEE/bin**), and will be named simply **core**. In this situation, the majority of **/IN/service_packages** will need to be checked for core files.

```
$ coreadm
     global core file pattern:
       init core file pattern: core
            global core dumps: disabled
       per-process core dumps: enabled
      global setid core dumps: disabled
 per-process setid core dumps: disabled
     global core dump logging: disabled
```

Single core location

However, if configured as in this example, all core files will be written to one central location (often on a separate mount point). In this situation, only one directory/mount needs to be checked.

This can also reduce the risk of an important mount point getting filled up with core files.

```
$ coreadm
     global core file pattern: /var/crash/core-%n-%p-%f
     global core file content: default
       init core file pattern: core
       init core file content: default
            global core dumps: enabled
       per-process core dumps: disabled
      global setid core dumps: enabled
 per-process setid core dumps: disabled
     global core dump logging: enabled
```

Diagnostic information

Processes that core can be a risk to the platform for many reasons, and should be dealt with as quickly as possible.

In general they indicate a software fault that will require investigation by Oracle Engineering, so it is important to collect the following diagnostic information:

Gdb backtrace

In order for Oracle Engineering to investigate a core file, the most important piece of information (apart from the core itself) is the gdb backtrace.

Follow these steps to collect the backtrace.

Step	Action

1	If not possible from the filename itself, determine what process created the core, using the file command.
	`$ file core`
	`core: ELF 64-bit LSB core file, x86-64, version 1 (SYSV), SVR4-style, from '/IN/service_packages/ACS/bin/slee_acs'`

Step	Action
2	Open the core using gdb, with the original binary and the core file as arguments.

Note: The exact binaries and libraries that generated the core file are required. If the product version has changed, it is unlikely gdb will be able to interpret the core correctly.

```
$ gdb /IN/service_packages/ACS/bin/slee_acs core

GNU gdb (Red Hat Enterprise Linux) 14.2-3.0.1.el9
Copyright (C) 2023 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-redhat-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from slee_acs...

warning: Can't open file /usr/lib64/libgcc_s-11-20231218.so.1 during file-backed
mapping note processing
[New LWP 473485]

warning: Build-id of /lib64/libstdc++.so.6 does not match core file.
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib64/libthread_db.so.1".
Core was generated by `/IN/service_packages/ACS/bin/slee_acs'
```

Result: Eventually you will be presented with the most recent frame of the core, the signal which ended the process, and a (gdb) prompt.

```
Program terminated with signal 10, Bus error.
#0  0xfe2d6328 in _smalloc () from /lib/libc.so.1
(gdb)
```

3	To view all frames in the core, initiate a summary backtrace by typing `bt` at the prompt, see *Example summary backtrace* (on page 83).

```
(gdb) bt
```

4	To view all frames and all their information in the core, initiate a full backtrace by typing `bt full` at the prompt, see *Example full backtrace* (on page 84).

```
(gdb) bt full
```

Note: This information will need to be provided to Oracle Support for further investigation.

Example summary backtrace

Initiate a summary backtrace by typing `bt` at the prompt, all frames in the core will be shown:

```
 (gdb) bt
#0  0xfe2d6328 in _smalloc () from /lib/libc.so.1
#1  0xfe2d639c in malloc () from /lib/libc.so.1
#2  0xfef63450 in operator new (sz=4) at new_op.cc:48
#3  0xfd318fa4 in cmn::escher::Array::push_back (this=0x18c1ce8, val=@0xffbfd030) at
cmnEscherEntry.hh:229
#4  0xfdc55248 in ccs::Message::CDR::appendFromString (this=0xffbfd0d8, fields=
        {static npos = 4294967295, _M_dataplus = {<allocator<char>> = {<No data fields>}, _M_p =
0x18c08c4 "CLI=10101010101|ACS_CUST_ID=12|PC_AC=1|PC_PRC=1|TZ=NZ|PC_SCD=D07"}, static
_S_empty_rep_storage = {0, 0, 0, 0}}) at /volB/DEV_BASE/nondebug/CCS/include/ccsMessage.hh:1581
#5  0xfdd030ac in fox::ExtendedWalletUpdate::doAction (this=0x1a13cb0, request=@0x1,
responseRequired=@0xffbfeb40, actionResponse=0x29c00,
```

```
    context=@0xffbfd0e0, serviceContext=@0x19eb4b8, parms=@0xffbfd2b0) at /opt/gcc-
3.2.3/include/c++/3.2.3/bits/stl_alloc.h:664
#6  0xfdc47fac in fox::FOXActionHandler::doAction (this=0x1a13cb0, request=@0xffbfeb40,
responseRequired=@0xffbfd21f, actionResponse=0x1a8c890,
    context=@0x19eb4b8, parms=@0xffbfd2b0) at FOXActionHandler.cc:1891
#7  0xff283128 in acsActionsAPI::ActionHandler::doAction (this=0x1a13cb0, parms=@0xffbfd2b0) at
acsActionHandler.cc:271
#8  0x000e7df4 in acsChassisInvokeAPluggableAction (event=0xffbfeb38, context=0x1a8c880,
actionStack=0x18c6588, result=0x1a8c888, callEnded=0xffbfd404,
    waitingForExternal=0xffbfd400, logErrorIfNotFound=1) at acsPluggableChassisAction.cc:358
#9  0x000e7670 in acsChassisInvokePluggableAction (event=0xffbfeb38, context=0x1a8c880,
actionStack=0x18c6588, result=0x1a8c888, callEnded=0xffbfd404,
    waitingForExternal=0xffbfd400) at acsPluggableChassisAction.cc:253
#10 0x00076a38 in acsSLEEChassis_t::doAction (this=0x18c6580, action=@0xffbfeb38,
actionYields=@0xffbfe957, actionExpectsResponse=@0xffbfe956)
    at acsChassis.cc:3854
#11 0x000723c4 in acsSLEEChassis_t::processCall (this=0x18c6580, context=0x1a8c880) at
acsChassis.cc:2498
#12 0x0006f9c8 in acsSLEEChassis_t::main (this=0x18c6580) at acsChassis.cc:1822
#13 0x0005b3c8 in main (argc=1, argv=0xffbff80c) at slee_acs.cc:134
```

Example full backtrace

Initiate a full backtrace by typing `bt full` at the prompt; all frames and all information contained in them will be shown. This can sometimes be many pages, and can sometimes result in endless junk information - collect as much as appears useful. The example below causes gdb to crash after the 5th frame:

```
 (gdb) bt full
#0  0xfe2d6328 in _smalloc () from /lib/libc.so.1
No symbol table info available.
#1  0xfe2d639c in malloc () from /lib/libc.so.1
No symbol table info available.
#2  0xfef63450 in operator new (sz=4) at new_op.cc:48
        p = (void *) 0x4
#3  0xfd318fa4 in cmn::escher::Array::push_back (this=0x18c1ce8, val=@0xffbfd030) at
cmnEscherEntry.hh:229
        this = (Entry * const) 0x18c1ce8
        this = (class ArrayImpl * const) 0x18c1ce8
        val = (const Map &) @0xffbfd030: {pimpl = {rep = 0x0}}
#4  0xfdc55248 in ccs::Message::CDR::appendFromString (this=0xffbfd0d8, fields=
        {static npos = 4294967295, _M_dataplus = {<allocator<char>> = {<No data fields>}, _M_p =
0x18c08c4 "CLI=10101010101|ACS_CUST_ID=12|PC_AC=1|PC_PRC=1|TZ=NZ|PC_SCD=D07"}, static
_S_empty_rep_storage = {0, 0, 0, 0}}) at /volB/DEV_BASE/nondebug/CCS/include/ccsMessage.hh:1581
        field = {pimpl = {rep = 0x1a25af0}}
        key = {static npos = 4294967295, _M_dataplus = {<allocator<char>> = {<No data fields>},
_M_p = 0x1a80f94 "PC_SCD"}, static _S_empty_rep_storage = {
    0, 0, 0, 0}}
        val = {static npos = 4294967295, _M_dataplus = {<allocator<char>> = {<No data fields>},
_M_p = 0x1aa7d3c "D07"}, static _S_empty_rep_storage = {0,
    0, 0, 0}}
        cdrEntry = {static npos = 4294967295, _M_dataplus = {<allocator<char>> = {<No data
fields>}, _M_p = 0x1a80eec "PC_SCD=D07"},
  static _S_empty_rep_storage = {0, 0, 0, 0}}
        equals = 4290760752
        start = 4290760768
        end = 64
#5  0xfdd030ac in fox::ExtendedWalletUpdate::doAction (this=0x1a13cb0, request=@0x1,
responseRequired=@0xffbfeb40, actionResponse=0x29c00,
    context=@0xffbfd0e0, serviceContext=@0x19eb4b8, parms=@0xffbfd2b0) at /opt/gcc-
3.2.3/include/c++/3.2.3/bits/stl_alloc.h:664
        cdr = {<Array> = {pimpl = {rep = 0x1a28d40}}, <No data fields>}
        parms = (acsChassisActionParms &) @0x1: <error reading variable>
        ewur = (class ExtendedWalletUpdateRequest *) 0xffbfeb40
        balanceInfoArray = {<Array> = {pimpl = {rep = 0x1a28cb8}}, <No data fields>}
        addBalanceInfoArray = true
        sbbia = (class SmallBalanceBucketInfoArray
Segmentation Fault (core dumped)
```

Memory leaks

While monitoring the platform, it may be determined that a certain process is constantly increasing in memory, indicating a memory leak.

Memory leaks can be a great risk to the platform, as other processes will struggle to run if the machine does not have enough free memory. In low memory situations the OS will start paging information in and out of memory, causing a performance impact, and system instability.

A slow leak may pose little danger to the platform; however, it is prudent to investigate sooner rather than later. In general leaks indicate a software fault that will require investigation by Oracle Engineering, so it is important to collect the following diagnostic information as soon as possible:

Diagnosing Memory Libraries

Follow these steps to check the memory libraries:

Step	Action
1	Log in as the root user.
2	Open the startup script. Add the following entries: **`MALLOC_CHECK=3`** **`export MALLOC_CHECK_`** **Result:** The process will abort with a core file when a memory check fails.
3	Log out of the root user.

Log files

All NCC processes write to their own log file, usually **/IN/service_packages/<Product>/tmp/***Process***.log**.

They will also write errors to the syslog, which generally has a longer retention period than log files. Log files are maintained by smsLogCleaner, which runs from each user's crontab using configuration in **/IN/service_packages/***Product***/etc/logjob.conf** usually once per hour.

Logs are archived to **/IN/service_packages/***Product***/tmp/archive/** and usually kept for seven days (configurable on the command line).

When a process is put in debug, this extra information is written to the log file only.

Note: Files archived by smsLogCleaner can have their names changed.

Debug

All NCC processes contain debug flags, which can be used to collect useful diagnostic information in the event of issues.

This is done in two main ways:

1 by specifying debug flags in the startup script - which results in debug for all processing as long as the process is up.
2 by setting tracing parameters inside configuration files.

The first is available to all NCC processes, the second to a select few traffic handling applications which require more targeted debugging.

Startup flags

After locating the process startup script, debug flags can be specified via environment variable (debug statement highlighted):

```
$ vi slee_acs.sh
#!/usr/bin/ksh
DEBUG=all,-COMMON_escher,-COMMON_escher_detail,-COMMON_FD,-COMMON_Utils,-slee_api
export DEBUG
exec /IN/service_packages/ACS/bin/slee_acs  >>
/IN/service_packages/ACS/tmp/slee_acs.log 2>&1
```

The flags available differ by process, and generally Oracle Support will advise the flags required. DEBUG=all covers all debug defined in the process, but will be quite verbose so should be limited.

Flags can be subtracted from "all" or individual flags specified.

Note: You can change the time zone for debug message timestamps by setting the environment variable in each associated startup script. Example:

```
DEBUG_TZ=America/Costa_Rica

export DEBUG_TZ
```

Available flags

To find out all the options available to a specific process, use the strings command along with grep.

For example type:

```
$ strings slee_acs | grep cmnDebug_FLAG
```

Result: All the flags available are listed.

```
cmnDebug_FLAG_Engine
cmnDebug_FLAG_Chassis
cmnDebug_FLAG_ACS_Chassis_CdrWrite
cmnDebug_FLAG_slee_acs
cmnDebug_FLAG_misc
cmnDebug_FLAG_COMMON_Utils
cmnDebug_FLAG_COMMON_Utils_cmnUnit
cmnDebug_FLAG_acsChassisSLEE
cmnDebug_FLAG_acsNOA
cmnDebug_FLAG_acsAWOL
cmnDebug_FLAG_acsCommon
cmnDebug_FLAG_acsCdr
cmnDebug_FLAG_Config
cmnDebug_FLAG_ConfigFileImpl
cmnDebug_FLAG_cmnPrefixTree
cmnDebug_FLAG_COMMON_cmnTime
cmnDebug_FLAG_cmnAssert
cmnDebug_FLAG_ACS_NotifIF
```

Note: The cmnDebug_FLAG_ prefix part is assumed by debug so can be left off when configuring the Debug command.

Flags to avoid

The following flags are used by the majority of processes, and result in a lot of debug.

They are recommended to be removed unless otherwise requested.

- COMMON_escher[_detail]
- COMMON_FD
- COMMON_Utils
- slee_api

Selective tracing

Selective debug is available to some of the more important real-time traffic handling processes. These include:

- slee_acs
- beVWARS
- xmsTrigger

In each case, a configurable tracing section contains a list of criteria for tracing (A-party and B-party for slee_acs, walletid for beVWARS), and will temporarily switch to debug for the duration of the triggering event.

Configuration can be made in **eserv.config** in the `tracing{}` section of the process, which is explained in full detail in the technical guides.

Once set, the process can be sent a SIGHUP signal to re-read its configuration, including the tracing section.

Tracing example

For example, here is an ACS tracing{} section for slee_acs:

```
tracing = {

    # Is tracing enabled? (default false)
    enabled = true

    # Originating Addresses that we want to trace
    origAddress = [
        "12345"
    ]

    # Destination Addresses that we want to trace
    destAddress = [
        "12345"
    ]

    # What debug level should the tracing be at?
    traceDebugLevel = "all"

}
```

xmsTrigger tracing

xmsTrigger tracing is set in the same fashion; however the resulting information goes to a separate file **xmsTrigger.trc**, does not contain debug, but does capture all the major decision points in a transaction.

Trace points are defined as:

Input

1 Message received from network With which addresses?
2 Message decoding information
   - Do we allow alternate delivery?
   - Which protocol version is this?
   - What was the message text (if showPrivate)?
3 Message passed to Messaging Manager
   - Result from ParentContext::handleSMSubmit?
4 Response received from MM
5 Response sent to network

Output

1 SMSubmit received from Messaging Manager
   - Is the delivery type SME or MC?
   - Do we need to consult a third party (for example, HLR) for any reason?
   - What are the addresses involved?
2 Outgoing encoding information
   - Which protocol version are we using?
3 Message sent to network
4 Response received from network
5 Response sent to Messaging Manager

Snoop traces

When dealing with issues related to real-time traffic handling, it is imperative to have reference snoop traces to observe the behaviour of the NCC software at the network/signalling level.

This information allows analysis of incoming messages, the responses sent back and the timing. Each standard is thoroughly documented and must conform to the appropriate specifications.

Snoop traces allow there to be no uncertainty about the conversation between the NCC platform and external components.

Running a snoop trace

Snoops are initiated as the root user. Command line arguments give the user a fair amount of control over what gets collected; from the interface to the port and transport protocol.

At a rudimentary level, snoop can be instructed to display all incoming traffic for an interface. However, it is more useful to first determine what traffic is required (the more detail the better) and save to a file for analysis in a trace interpreter.

To see a list of all the snoop command line parameters, type:

```
$ man snoop
```

This gives a full list, with definitions.

Snoop example

In this example, diameterControlAgent has a handle on the local address 172.21.153.142 on port 3868. Using ifconfig, this is shown to be on interface e1000g1.

Note: Network Connectivity Agents (NCAs) commonly use more than one interface for receiving/sending information. There are failover and loadsharing scenarios where this is required. The groupname specified will sometimes indicate the type of traffic, for example, "SIG-A" and "SIG-B" shows that more than one interface is used for SIGTRAN.

First, determine the interface the target process is attached to. This can be achieved by checking the output of ifconfig, inspecting the process with pfiles and cross-checking the results as highlighted:

```
$ ps -ef | grep diameterControlAgent

acs_oper   160     1   0   Oct 20 ?         251:34 diameterControlAgent
$ pfiles 160 | grep sock

        sockname: AF_UNIX /tmp/dcaIf-0.0.112.20101020123758
        sockname: AF_INET 0.0.0.0  port: 3868
        sockname: AF_INET 172.21.153.142  port: 3868
        sockname: AF_INET 172.21.153.142  port: 3868
$ ifconfig -a

lo0: flags=2001000849<UP,LOOPBACK,RUNNING,MULTICAST,IPv4,VIRTUAL> mtu 8232 index 1
        inet 127.0.0.1 netmask ff000000
e1000g0: flags=1000843<UP,BROADCAST,RUNNING,MULTICAST,IPv4> mtu 1500 index 2
```

```
        inet 172.21.153.82 netmask ffffffc0 broadcast 172.21.153.127
        groupname mgmt
e1000g0:1: flags=9040843<UP,BROADCAST,RUNNING,MULTICAST,DEPRECATED,IPv4,NOFAILOVER> mtu 1500
index 2
        inet 172.21.153.80 netmask ffffffc0 broadcast 172.21.153.127
e1000g1: flags=1000843<UP,BROADCAST,RUNNING,MULTICAST,IPv4> mtu 1500 index 3
        inet 172.21.153.142 netmask fffffffe0 broadcast 172.21.153.159
        groupname chrg
e1000g1:1: flags=9040843<UP,BROADCAST,RUNNING,MULTICAST,DEPRECATED,IPv4,NOFAILOVER> mtu 1500
index 3
        inet 172.21.153.140 netmask fffffffe0 broadcast 172.21.153.159
e1000g2: flags=1000843<UP,BROADCAST,RUNNING,MULTICAST,IPv4> mtu 1500 index 4
        inet 172.21.5.100 netmask ffffff00 broadcast 172.21.5.255
        groupname sig
e1000g2:1: flags=9040843<UP,BROADCAST,RUNNING,MULTICAST,DEPRECATED,IPv4,NOFAILOVER> mtu 1500
index 4
        inet 172.21.5.104 netmask ffffff00 broadcast 172.21.5.255
e1000g3: flags=1000843<UP,BROADCAST,RUNNING,MULTICAST,IPv4> mtu 1500 index 5
        inet 172.21.205.27 netmask ffffff00 broadcast 172.21.205.255
nxge0: flags=9040843<UP,BROADCAST,RUNNING,MULTICAST,DEPRECATED,IPv4,NOFAILOVER> mtu 1500 index 6
        inet 172.21.153.81 netmask ffffffc0 broadcast 172.21.153.127
        groupname mgmt
nxge1: flags=9040843<UP,BROADCAST,RUNNING,MULTICAST,DEPRECATED,IPv4,NOFAILOVER> mtu 1500 index 7
        inet 172.21.153.141 netmask fffffffe0 broadcast 172.21.153.159
        groupname chrg
nxge2: flags=9040843<UP,BROADCAST,RUNNING,MULTICAST,DEPRECATED,IPv4,NOFAILOVER> mtu 1500 index 8
        inet 172.21.5.101 netmask ffffff00 broadcast 172.21.5.255
        groupname sig
```

Information level of detail

To collect all information on the example interface, we can use the `-d` argument along with `-o` to get an output snoop file for our interpreter to use:

```
$ snoop -d e1000g1 -o diameterControlAgent.snoop
```

However, to target the snoop even more, we can also restrict to port 3868 using the **-c** argument.

```
$ snoop -d e1000g1 -c tcp port 3868 -o diameterControlAgent.snoop
```

Note: tcp is assumed as Diameter is a tcp protocol.

To run a snoop for an extended period of time, it can be called with `nohup` or suffixed with `&` to have it run in the background. In this situation it is recommended to also use the `-q` argument, which suppresses the packet count.

Snoop interpreter

Once a snoop has been collected, an interpreter can be used to view the packets in a graphical interface.

Wireshark is one such widely used protocol analyzer, and contains plugins for decoding many telephony protocols, including:

- INAP
- Camel
- MAP
- Diameter.

Wireshark contains many useful features, which are outside of the scope of this document. In general, it will work quite well out of the box, automatically recognizing and decoding protocols without need for special configuration. For more information, see the Wireshark website www.wireshark.org.

Process failure

You can check whether a process is restarting using the SMS Alarms subsystem.

Processes raise alarms when they are stopped or started. The alarms include:

- Their name
- The time the alarm was logged
- Some other information about why the event may have occurred

Further information about the specific alarm can be found in the application's alarms guide.

Alarms can be accessed from the:

- Syslog on the local machine and the SMS(s). For more information, see *SMS Technical Guide*.
- **Alarms** tab in the SMS Alarms Management screen. For more information, see *SMS User's Guide*.

Checking installed packages

To check the details of an installed package, use the `pkginfo` command.

Example command: `pkginfo -l smsSms`

Example output: This is an example of the output of the example command above.

```
   PKGINST:  smsSms
      NAME:  Oracle smsSms
  CATEGORY:  application
      ARCH:  sun4u
   VERSION:  3.1.0
    VENDOR:  Oracle
    PSTAMP:  smsNode20041020104925
  INSTDATE:  Oct 20 2004 13:15
     EMAIL:  support@oracle.com
    STATUS:  completely installed
     FILES:      348 installed pathnames
                  39 directories
                  89 executables
              152448 blocks used (approx)
```

For more information about the `pkginfo` utility, see the system documentation.

Checking access to Oracle database

A number of services and functions rely on access to the Oracle database. To check that the Oracle database is available to a service, check the following:

1 Use sqlplus to check that you can log in to the Oracle database with the username and password the service is using to connect.

   Example command: `sqlplus smf/smf`

2 Where the tables required for a service are known, use SQL queries to check that:
   - The tables exist
   - The tables have appropriate content

For more information about SQL queries, see the Oracle documentation.

Checking network connectivity

Network connectivity will affect any process which requires communication between two different network addresses.

Network connectivity should support ssh sessions between the two machines experiencing the problem.

If you can open an ssh session between the two machines, check the following before contacting Level 1 support with details:

- If the address of either of the machines specified in the Node Management screens is a hostname, check that the hostnames used in the ssh sessions are the hostnames specified in the Node Management screen.

If you cannot ssh, check the following before contacting Level 1 support with details:

- Check that the hostname is resolving correctly in the DNS.
- Check that the physical network connection is working correctly.
- Check that the inetd and sshd are running.
- Check that sshd is listening on the expected port.
- Check that the smf_oper and acs_oper accounts are not locked, and that the username and password combinations being used are correct.

Replication

Replication may be failing for the following reasons:

- ssh keys have not been correctly set up between origin and destination machines.
- The destination node has been incorrectly set up in the Node Management screens of the SMS Java screens.
- Oracle is not running correctly.
- A new replication.cfg file has not been created after a change.
- replication.cfg may not be successfully copying to the destination machine (an error should display when the **Create Config File** button on the Node Management screens is clicked).
- The partition on the destination machine where the data is being replicated to may be full.
- The updateLoader on the destination machine may be running incorrectly.
- The destination database may be substantially out of sync with the SMF. Run a resync.

Appendix A

NCC Directory Structure and Contents

Component directory structure and contents

This table lists the product directory structure for each component product. The default installation directory for each product is:

/IN/service_packages/product_home/

Each component product is installed in this directory, for example, **/IN/service_packages/ACS/** is the product home directory for the ACS product.

Directory	Description
/IN/service_packages/product_home/	Product Home directory.
/IN/service_packages/product_home/bin	Product Binary executables and .sh files.
/IN/service_packages/product_home/tmp	Product log files.
/IN/service_packages/product_home/etc	Product configuration files.
/IN/service_packages/product_home/lib	Product library executable files.
/IN/service_packages/product_home/db	Product database installation scripts.
/IN/service_packages/product_home/cdr	Product EDR files.

Component Product Directories and Description

This table lists the set of component (product home) directories installed as part of an NCC install and which conform to the product directory structure as described in the Component directory structure and contents.

Notes:

- Not all component products exist on each NCC server.
- Not all sub-directories will exist for each component product.
- The component list will depend on the specific NCC installation and will most likely be a sub-set of all NCC components.

Component	Description
OSD	Open Services Development
PI	Provisioning Interface
DAP	Data Access Pack
ACS	Advanced Control Services
SMCB	Short Message Charging
USSD	USSD Roaming Application
RAP	Roaming Application Part (Camel Roaming)
VSSP	Virtual SSP
SMSC	SMS Interface
SCA	Session Control Agent
IS41	U-CA-IS41 (CDMA)

Component	Description
DCA	Diameter Control Agent
DCD	Diameter Control Driver
SLEE	Service Logic Execution Environment
CCS	Charging Control Services
XMS	Messaging Manager application
SMS	Service Management System
E2BE	Voucher and Wallet Server
RIMS	Routing Information for Mobile Services
LCP	Location Capabilities Pack