Oracle® Communications Convergent Charging Controller

Session Control Agent Technical Guide

Release 15.0.0

October 2023

Copyright

Contents

About This Document v
Document Conventions vi

Chapter 1

System Overview 1

Overview 1
What is the Session Control Agent? 1
Deploying the SCA 4
Configuration Overview 5
ENUM 6
ESC 7
Next Hop Processing 9
URI / E.164 Translation 10
Parallel Hunting 10
Presence and Availability 11
PRACK Support 11
Custom SIP Headers 11
LAN Redundancy 12
Local Address Redundancy 13
Remote Address Redundancy 14
Normalization and Denormalization 17
CLIP and CLIR 18
Feature Nodes 18

Chapter 2

Prepaid Platform and NGN Integration 21

Overview 21
Key Functionality 21
Prepaid Platform Heartbeating 22
Media Description Retrieval from SDP 30
Call ID retrieval 31

Chapter 3

Scenarios 33

Overview 33
Call Forwarding 33
Call Redirection 36
Callback 36
SIP Error Response 40

Chapter 4

Configuration 41

Overview 41
Configuration Overview 41
sca.config Configuration 42
SCA Configuration 43
ESC Configuration 77
Parser Configuration 84

Presence Configuration 85
acs.conf Configuration 87
SLEE.cfg Configuration 88
Configuring EDR Collection 88
Configuring IN Call Model Triggers 89

Chapter 5

NHP Configuration 93

Overview 93
NHP Configuration File 93
NHP Rule Definition 94
NHP Commands 96
NHP Host Definition 99

Chapter 6

Background Processes 101

Overview 101
sca 101

Chapter 7

Tools and Utilities 103

Overview 103
registrar 103
remoteCommanderUser 104
SCA Remote Commander 105
Statistics 108
SCA EDRs 110

Chapter 8

About Installation and Removal 113

Overview 113
Installation and Removal Overview 113
Checking the Installation 113

About This Document

Scope

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

Audience

This guide was written primarily for system administrators and persons installing, configuring and administering the Session Control Agent application. However, sections of the document may be useful to anyone requiring an introduction to the application.

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:

- *Advanced Control Services Technical Guide*
- *Control Plan Editor User's Guide*
- *Service Management System Technical Guide*
- *Service Management System User's Guide*
- *Service Logic Execution Environment Technical Guide*

Document Conventions

Typographical Conventions

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

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

Chapter 1

System Overview

Overview

Introduction

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

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

In this Chapter

This chapter contains the following topics.

What is the Session Control Agent? 1
Deploying the SCA 4
Configuration Overview 5
ENUM 6
ESC 7
Next Hop Processing 9
URI / E.164 Translation 10
Parallel Hunting 10
Presence and Availability 11
PRACK Support 11
Custom SIP Headers 11
LAN Redundancy 12
Local Address Redundancy 13
Remote Address Redundancy 14
Normalization and Denormalization 17
CLIP and CLIR 18
Feature Nodes 18

What is the Session Control Agent?

Overview

The Session Control Agent (SCA) is a highly available and scalable SIP transparent back-to-back user agent (B2BUA), redirect server, proxy server and registrar. It provides support for SIP/SIMPLE messaging and integrated triggering capabilities for Oracle Communications Convergent Charging Controller and third-party applications.

The SCA enables real-time charging, instant messaging and personal mobility in SIP-based next generation networks (IETF/ETSI NGNs) and in the IP Multimedia Subsystem (3GPP IMS, 3GPP2 MMD).

It can be integrated into Convergent Charging Controller's Prepaid Charging solutions based on Charging Control Services (CCS), with on-line charging interfaces to the Convergent Charging Controller Voucher and Wallet Server and to third party real-time billing systems.

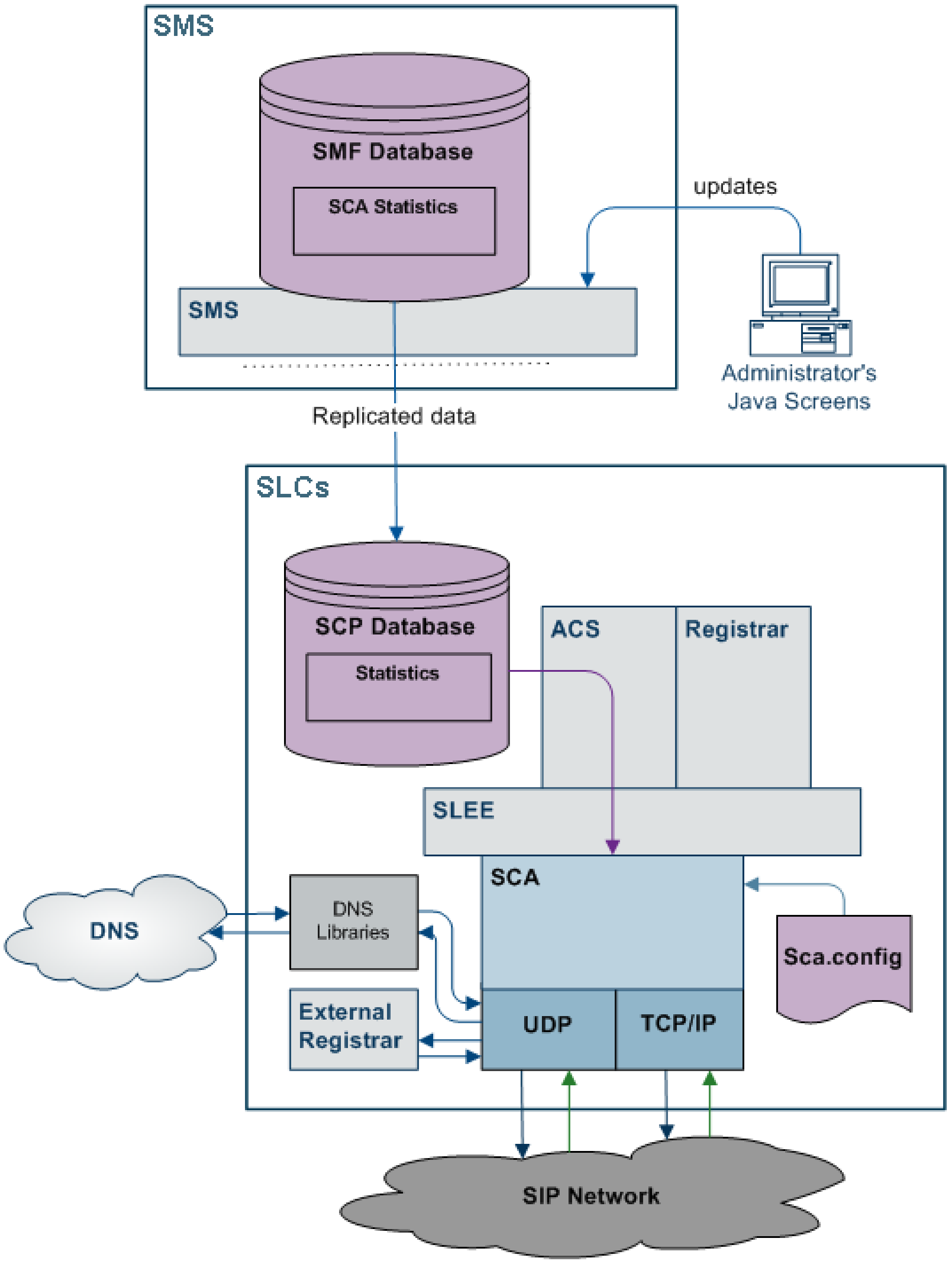
Features

The SCA has the following capabilities. It can:

- Send and receive SIP requests and responses to, and from:
  - Remote SIP clients and servers, over TCP/IP or UDP
  - SLEE interfaces or applications
- Query the DNS to retrieve IP addresses for hosts, and SRV records for domains
- Query custom-built PAMs through a SLEE PAM interface that translates SIP register requests
- Query SIP based PAMs directly
- Trigger IN applications such as ACS
- Provide next hop processing through a set of predefined rules
- Provide support for parallel hunting for connect requests received through ACS
- Send queries received through ACS to a remote presence server for subscriber presence and availability information
- Publish subscriber presence and availability information, received through ACS, to a remote presence server
- Provide support for PRACK
- Act as a B2B-UA when forwarding INVITE requests
- Support local and remote LAN redundancy
- Normalize and denormalize all numbers passed to and from the IN systems
- Handle calling line identification presentation based on values received in the incoming INVITE request
- Restrict calling line identification by setting the appropriate fields within the IDP

SCA components

Here is a diagram of the SCA components.



Deploying the SCA

B2BUA or proxy

The SCA can be deployed in either transparent back-to-back user agent (B2BUA) mode or proxy mode. This can be configured globally using the b2bua configuration option. When set to true, all calls will be handled in B2BUA mode. When set to false, calls will be handled in proxy mode unless the B2B-UA flag has been added to individual rules in the **rules.nhp** file.

Note: By default the SCA handles all calls in B2BUA mode.

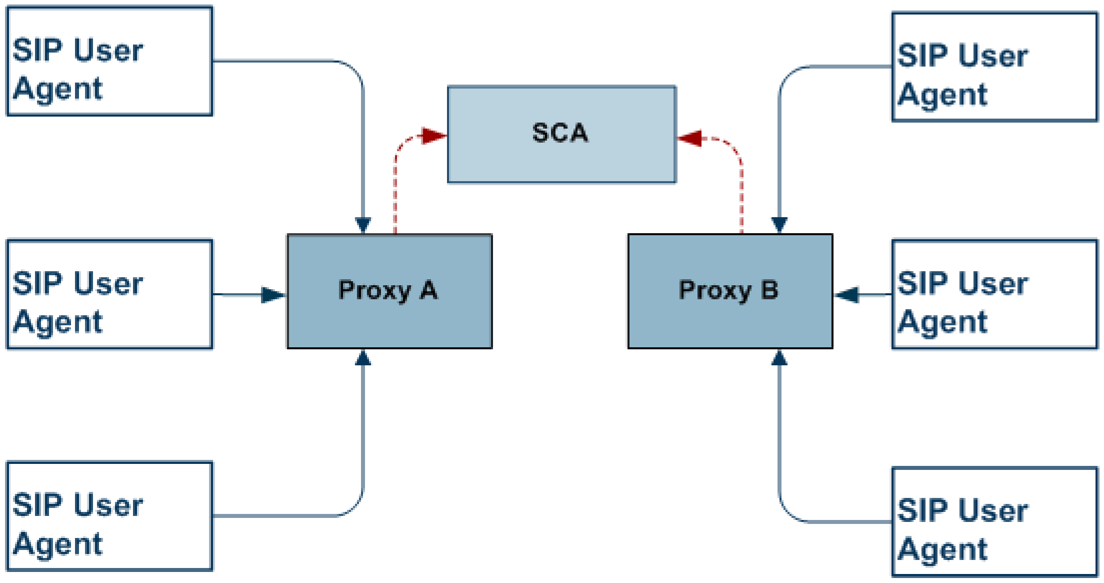
SIP proxy deployment

A SIP proxy is used to forward a request from one physical location to another. This is the recommended mode of operation for one of the following:

- The source and destination of SIP traffic are physically separate locations
- The SCA is running in standalone mode or is using a very simple control plan (for example, Start->Attempt-Terminate->End)
- The SCA is required to "get out of the loop" after forwarding the first request. For example, after successfully forwarding the initial request the SCA will be bypassed and all subsequent requests within a call will be sent directly between the endpoints.

SIP proxy deployment diagram

Here is an example SIP proxy deployment diagram. In the diagram a SIP user agent represents any SIP capable device such as a softphone.



B2BUA deployment

A transparent B2BUA controls call legs independently allowing it to perform complex call scenarios and to forward requests to the same physical location that originated the call. This type of deployment is recommended if:

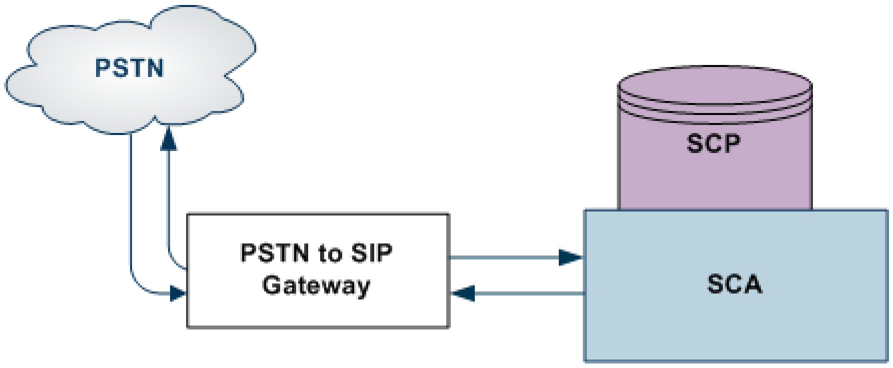
- The source and destination of SIP traffic is likely to be the same physical entity (for example, a PSTN to SIP gateway)
- Complex control plans are being used that may redirect a call to a different destination, such as an IVR. This may occur during the call or after it completes

- Calls may be initiated by ACS (the call-back scenario)
- The SCA will be performing parallel hunting

Note: If the SCA is deployed in proxy mode and one of the above scenarios is triggered, the remainder of the call will automatically be handled in B2BUA mode.

B2BUA deployment diagram

Here is an example SIP Transparent B2BUA deployment diagram.



Configuration Overview

Introduction

This section provides a brief overview of the mandatory and common configuration required to run the SCA. The SCA configuration file (**sca.config**) is installed in **/IN/service_packages/SCA/etc** with all the mandatory items configured. An example configuration file (**sca.config.all.example**) is also installed which includes all configuration options.

Note: For a full description of all the configuration options refer to *Configuration* (on page 41).

Mandatory configuration

This table describes the parameters in the **sca.config** file that must be configured for the SCA to work.

Parameter	Description
`local_ip`	This must be set to the IP address of the machine on which the SCA is running. The IP address will be inserted into all SIP requests sent by the SCA to allow responses to be routed back to the SCA.
`rem_comm_port`	If the remote commander is enabled, then it will listen on the specified TCP port.
`dns_server`	If the SCA is required to perform DNS lookups, then this must be set to a valid DNS server address. If a DNS server is not required, then this option can be left at the default value. Hostname to IP address mappings can be added to the **/etc/hosts** file. If the dns_check_files configuration option is enabled, then the SCA will read this file before attempting to query a DNS server.

Parameter	Description
`default_domain`	This defines the default domain name that will be appended to E.164 numbers returned from a control plan. The `trans_mapping` and `trans_behaviour` configuration options allow different domains to be appended depending on the prefix.

Common configuration

This table describes the common optional configuration decisions when deploying the SCA.

Configuration	Description
EDRs	By default the SCA logs EDRs to **/IN/service_packages/SCA/cdr/closed**. This can be disabled by setting the enableCDRs configuration option to false.
UDP or TCP	By default the SCA listens on TCP and UDP port 5060 for SIP traffic (this is configurable). UDP is the recommended transport protocol for SIP traffic.

General

The SCA includes functionality to handle network and gateway failure. For details, see *LAN Redundancy* (on page 12).

ENUM

Introduction

As part of the WaitingOnIN state, the SCA may be optionally configured to expect and look for results from an ENUM lookup. This information will be available within an outgoing CONNECT message, as:

- An AUS, typically an e.164 number with a leading '+' (for example: +441473289900)
- A URI, a string typically of the form: www.oracle.com
- One or more DNS NAPTR records, a NAPTR record contains a regular-expression based replacement rule and protocol for the initial calling AUS, to convert it to a URI
- An AUS, a URI, or a set of NAPTR records, dynamically, data will have its type identified by a flag value within the data

Encoding

This information is encoded into a profile block structure, and will be in a standard location.

Multiple URIs

Where multiple valid URIs are derived from ENUM NAPTR information, the SCA will use the parallel INVITE feature to send INVITES to all identified parties at the "same" time.

Using NAPTR records

It is possible to configure the SCA to use NAPTR records of a particular service type, for example, "E2U+SIP", and not use any other service type.

Dynamic data identification

Where the data type is dynamically identified, the data must be in the ENUM dynamic record format. This is the default for results from the ENUM Query, ENUM Call Out and NAPTR results from the ENUM Naptr Response node.

Supported expressions

When processing NAPTR records, the SCA will use very limited regular expression support. For example, a typical AUS replacement regular expression would be:

```
!^.*$!+441473289900@oracle.com!
```

Where '!' is a delimiter (which splits the string into two sections). This will replace any input AUS with the string: '**+441473289900@oracle.com**'.

Only full replacement expressions are supported, where the left-hand-side is: "^.*$" (as in the above example).

Unsupported expressions

In full NAPTR support, the following could be used:

"!^+441234(.*)$!+441473\1@oracle.com!"

This response contains a back-reference to information in the input AUS ('\1'). Such expressions are not supported.

ESC

Introduction

The Convergent Charging Controller SIP Chassis (ESC) component of the SCA provides the functionality for sending and receiving SIP requests and responses over the following transport mechanisms:

- UDP
- TCP/IP
- SLEE

The available transport mechanisms are dependent on the configuration defined for the ESC in **sca.config**. This part of the ESC is called the 'transport layer'.

Warning: TCP and UDP networking can be configured to be enabled or disabled. However, if they are both disabled, then the SCA will fail.

TCP networking

The ESC polls for incoming TCP connections on a configurable network address and port. For more information on how to configure TCP networking, see *ESC Configuration* (on page 77).

Once a connection has been accepted, the incoming data is read until the following information is received:

- Request line,
- Message header
- Message body

UDP networking

The ESC polls for incoming UDP datagrams on a configurable network address and port. It expects the SIP requests and responses to be contained in the same datagram. For more information on configuring UDP networking see *ESC Configuration* (on page 77).

SLEE networking

The ESC defines SLEE SIP request and SLEE SIP response events that allow the SCA to communicate with external SLEE interfaces and applications, such as the IN application or PAM interface.

The SCA polls the SLEE for events, and filters out anything that is not a SIP request or SIP response. These are then passed to the ESC for processing.

Request processing

Any incoming SIP message that does not start with the string "SIP/" is assumed to be a request. When a SIP request is received, the request line is checked to ensure it contains a method name, request URI and SIP version number (this must be 2.0).

The following SIP methods are supported. If the method name is anything else, then an error is reported:

- ACK
- BYE
- CANCEL
- INFO
- INVITE
- MESSAGE
- NOTIFY
- OPTIONS
- PRACK
- PUBLISH
- REGISTER
- SUBSCRIBE
- UPDATE

Message header

The message header for both SIP request and SIP responses must contain the following mandatory fields:

- To
- From
- Cseq
- Call-ID
- Max-Forwards
- Via

Request message headers that do not contain all the fields will result in an error. Response message headers that do not contain all the fields will also result in an error and will be ignored.

Note: Cseq number for all SIP messages during PRACK and REINVITE scenarios are calculated as below:

Prack Cseq numbers get sequentially incremented from INVITE_CSeq_Number+100 and re-INVITE Cseq numbers are sequentially incremented from just previous INVITE_CSeq_Number+200. BYE CSeq number are calculated based on previous_INVITE_CSeq_Number+199.

DNS

This table describes the types of resource record retrieved when the ESC queries the DNS.

Record Type	Description
SRV	Indicates which proxies to use for particular domain and transport protocols.
CNAME	Provides aliases for host redirection.
A	Provides the actual IP addresses for hosts.

Next Hop Processing

Introduction to Next Hop Processing

The SCA uses next hop processing (NHP) to determine what action to take for incoming SIP requests and partially processed SIP requests. The SCA also uses next hop processing when sending outbound SIP INVITE messages during third-party call control (3PCC) call setup.

The NHP is controlled by a set of rules contained in the **rules.nhp** configuration file. This file is read and preprocessed by the SCA at startup. For information about how to configure NHP rules, see *NHP Configuration* (on page 93).

NHP SIP Request Actions

The next hop processing can perform the following actions on the SIP request:

- Redirect it to an alternative host/domain
- Forward it to an alternative host/domain using load-balancing
- Send multiple INVITE requests when the destination routing address (DRA) for the incoming request contains a list of numbers
- Perform a location query and forward it to the location returned by the query. If multiple locations are returned, they are tried in order
- Convert it to CS1 InitialDP and send it to an external IN application
- Forward it to an external SLEE application
- Handle it internally

NHP SIP Invite Actions

The next hop processing can perform the following actions on the SIP Invite to improve integration with Oracle Communications Service Controller for third-party call control (3PCC):

- Insert custom Session Initiation Protocol (SIP) headers into INVITE messages during 3PCC setup.
- Insert a SIP header that contains content that was derived from a previously received SIP message.

NHP triggers

Next hop processing can be triggered on:

- A numerical address that starts with a given prefix in the From or To header field.

- *user@host*, or *user@domain* in the From or To header field, where the host or domain ends with a given suffix.
- *user@host*, or *user@domain* in the From or To header field, where the user starts or ends with a given prefix/suffix.
- *user@IPaddress* in the From or To header field, and the IP address starts with a given prefix.
- *user@host*, or *user@host* in the VIA header.
- A regular expression in the From or To header field.
- A regular expression in the body of the message.
- The status code in the header in the SIP response.
- Unconditionally.

Location query

Location query can be performed as part of next hop processing. The responses from the PAM regarding the location are stored in the location cache by sending one of the following:

- REGISTER SIP request to a remote SIP element, identified by its hostname or IP address
- REGISTER SIP event to a SLEE application or interface.

In both these cases the request URI determines where the request is actually sent.

Tip: For details on the registrar tool provided as part of the SCA see the *registrar* (on page 103).

URI / E.164 Translation

URI / E.164 translation

The SCA supports both URIs and E.164 numbers (as used in ACS). The ability to map URIs to E.164 numbers to facilitate conversion between the two types is provided. For information on mapping URIs to E.164 numbers see *SCA Configuration* (on page 43).

Parallel Hunting

Introduction

The SCA supports parallel hunting for connect requests, received through ACS, which contain a list of numbers in the destination routing address (DRA). It can be configured to send INVITE requests to multiple destinations. As soon as the call is answered by one of the destination parties, the call to all other parties is canceled.

Feature node

The Copy Hunting List feature node available in the ACS Control Plan Editor is used to obtain the list of numbers in the DRA.

For more information on ACS feature nodes and the Control Plan Editor, see *Feature Nodes Reference Guide* and *Control Plan Editor User's Guide*.

Presence and Availability

Introduction

Subscribers to the SCA can publish their presence and availability details to a remote presence server. The details can then be queried in order to influence routing of subsequent calls. For example, calls may be redirected to voicemail if the subscriber is busy.

Presence configuration

The possible locations and availabilities that a subscriber may have are defined in the **eserv.config** configuration file. See *Presence Configuration* (on page 85) for details.

Presence feature nodes

Requests to publish and query presence and availability details are sent to the SCA through ACS feature nodes. The SCA routes the requests to the remote presence server and returns the responses to the originating ACS feature nodes.

The following presence nodes are available in the ACS Control Plan Editor:

- Presence Branching
- Set Presence

For more information on ACS feature nodes and the Control Plan Editor, see *Feature Nodes Reference Guide* and *Control Plan Editor User's Guide*.

PRACK Support

PRACK support

The SCA can be configured to provide support for PRACK. When this is enabled, the SCA will send a standard SIP acknowledgment message for all provisional requests in the range 100 to 199.

For configuration details see *ESC Configuration* (on page 77).

Custom SIP Headers

About Custom SIP Header Support

The SCA provides the following features to support custom SIP headers for third-party call control (3PCC):

- The ability, through configuration, to send an empty SDP connection address of c=IN IP4 0.0.0.0 in the INVITE message of a 3PCC call-initiation scenario when `sca.call_init_use_reinvite = true`.

  You configure `call_init_send_empty_address` in the `sca` section of the **sca.config** file. For information, see *call_init_send_empty_address* (on page 76).
- The ability to insert custom Session Initiation Protocol (SIP) headers into INVITE messages during 3PCC setup. For information, see *PUT_HEADER Command* (on page 99).
- The ability to insert a SIP header that contains content that was derived from a previously received SIP message. See *Adding Inbound Header to Subsequent INVITE Messages* (on page 12).

For example, you could use these features to improve integration with Oracle Communications Service Controller for third-party call control.

Adding Inbound Header to Subsequent INVITE Messages

SCA supports the ability to insert into an outbound INVITE message a SIP header that contains content that was derived from a previously received SIP response.   For example, you can use this capability to set the value of the retrieved header in an arbitrary header in a subsequent 3PCC INVITE message.

SCA supports this capability with the following features of NHP rules:

- The STATUSCODE identifier in the `HOSTS` section of the NHP configuration file, **rules.nhp** that allows a command to be triggered based on a SIP response.
- The GET_HEADER command that allows a header to be retrieved from an inbound SIP response and stores the value in memory with a specified *key_name* identifier.
- The PUT_HEADER command that can then be used to populate the header of subsequent outbound INVITE messages with the value that the GET_HEADER command retrieved from the SIP response.

For more information about NHP configuration and NHP commands, see *NHP Configuration* (on page 93).

Example configuration: Adding inbound header to subsequent INVITE message

```
RULES = {
    "r1" = ( DEFAULT ) LOCATION ( SK = 200 );
    "registrar" = ( DEFAULT ) FORWARD ( SK = 200 );

    "trying" = ( DEFAULT ) GET_HEADER ( NAME = "x-wcs-encode-uri"
        ID = "ID_route" };
    "get_route" = ( DEFAULT ) PUT_HEADER ( NAME = "Route" ID =
        "ID_route" };
}

HOST * = {}
    STATUSCODE "183" = { "trying" }
    METHOD "INVITE" = { "get_route", "r1" }
    METHOD "REGISTER" = { "registrar" }
}
```

The receipt of a SIP 183 response triggers the `trying` rule, which stores the content of the `x-wcs-encode-uri` header in memory with a key of `ID_route`.

When sending the SIP INVITE message, the rule `get_route` is triggered, adding a `Route` header to the message, with its value set to the content that was stored in memory with the key value `ID_route`.

LAN Redundancy

Introduction

The SCA supports redundant LAN deployments for failover local addresses on the SLC or remote IP addresses to which it is directing traffic. The two types of LAN redundancy supported are:

- Local address redundancy: this enables the SCA to detect failure of local network interfaces and stop billing for calls if such failure occurs.
- Remote address redundancy: when initiating a call, this enables the SCA to detect LAN address failures of the remote SIP entities to which it directs traffic and to try alternate destination URLs.

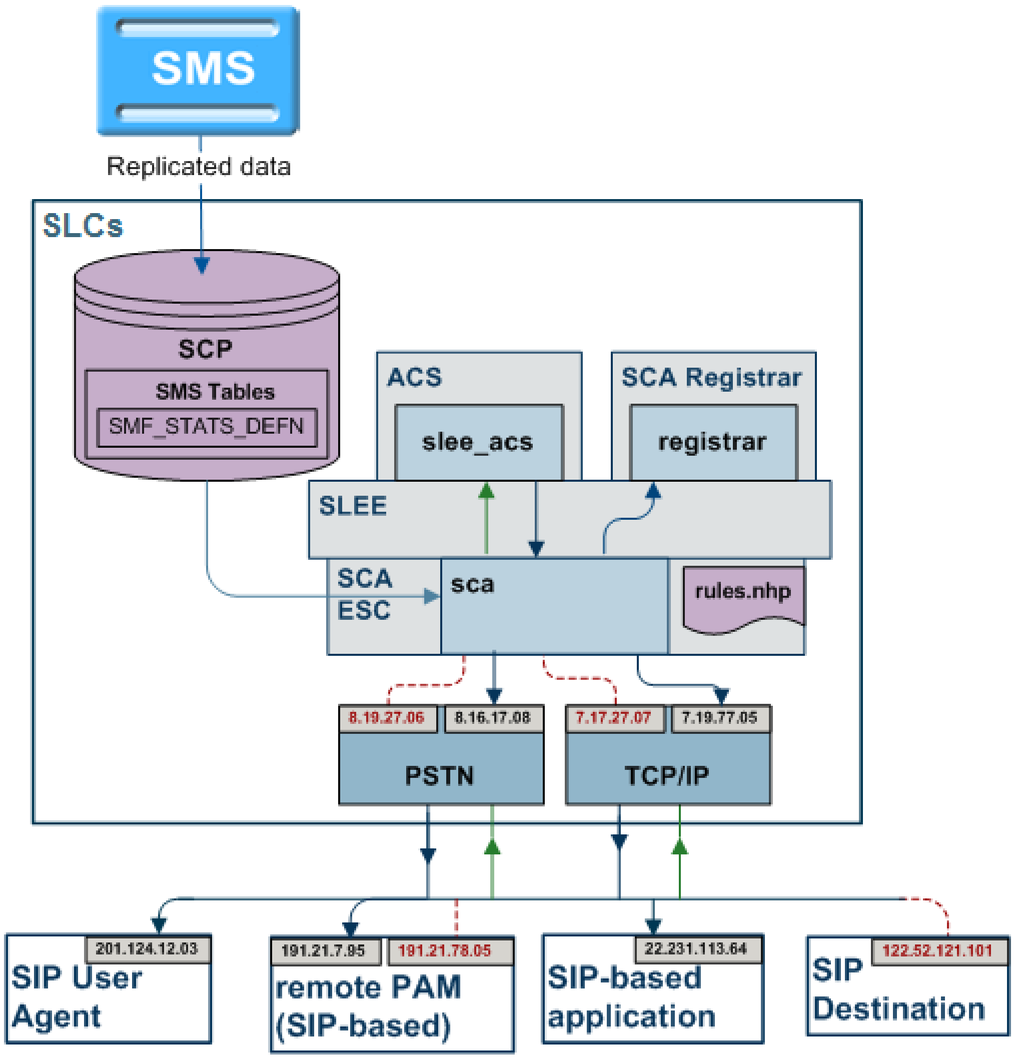
Note: These features are not a substitute for a fully redundant network architecture.

- If mid-call failover and redundancy in the mediastream is required, then the network on which the SCA is deployed should include switches and routers that can detect failure and send traffic along an alternate path.

- If the SCA is being used to handle billed calls, then you should configure SCA to send heartbeat messages to the gateways to detect remote gateway failure. For more information, see *Prepaid Platform Heartbeating* (on page 22).

Diagram

Here is an example of a network layout with redundant addresses.



Local Address Redundancy

Introduction

To support local address redundancy, the SCA constantly monitors the status of user-defined LAN addresses on the local SLC. It checks the operating system flags and determines if any local addresses have failed. If all interfaces have failed, the SCA will notify the IN to release any active calls.

Procedure

The detailed process is explained below:

1 Users can define a list of LAN addresses to be monitored in the SCA configuration file.
2 The SCA constantly monitors the LAN interface status flags on the defined addresses for LAN failure.
3 If all interfaces are marked as DOWN, the SCA will raise an alarm notification and the IN will release any active calls.
4 The SCA continues to monitor the interfaces.

Warning: If no interfaces are defined in the SCA configuration file, the SCA will not check for any failed local interfaces and behave as per normal.

Remote Address Redundancy

Introduction

The SCA also monitors for user-defined remote client addresses when it is initiating requests to remote SIP entities over the network.

The important features of remote address redundancy are:

- Load-balancing
- Transaction Timer
- Alternate Addresses

Process

The detailed process is explained below.

Stage	Description
1	Users can define a list of destination URLs to be monitored in the DNS entry or in the SCA Rules file.
2	The SCA sends a request message and retransmits it at intervals configured in the transaction timer.
3	The timer is canceled when an appropriate response is received before the timer expires.
4	If the timer expires and no response is received, the SCA will mark the interface as DOWN and notify the IN.
5	The SCA will send the message to the next alternate address provided in the User-defined list.
6	The SCA continue through the list until an address is successful or all URLs have been tried.
7	If all addresses fail, the SCA will notify the IN to release all active calls to the remote SIP entity and sends an error message to the A-party.

Note: If a single address is specified for a remote SIP entity, the SCA behaves as per normal.

Load balancing

Users specify a list of remote addresses for each SIP client in the NHP file or the DNS server. The SCA load balances requests between the defined IP addresses using a round robin approach. Once an address is detected as failed, it will be cached for a specified period to avoid sending subsequent messages to the failed address.

The load balancing feature can be optionally disabled. When disabled, all subsequent messages are sent to the first available address. If the load balancing behavior is enabled, excluding failed addresses, the messages will be sent to each address in the list in turn.

Alternate addresses

In the event of a remote address failure, the SCA will try alternate destination IP addresses defined for that client in the list of alternate destination URLs specified in a DNS entry or in the SCA Rules file. When the transaction timer expires the SCA will send the message to the next URL in the provided list. The SCA will continue through the list until an address is successful or all URLs are exhausted.

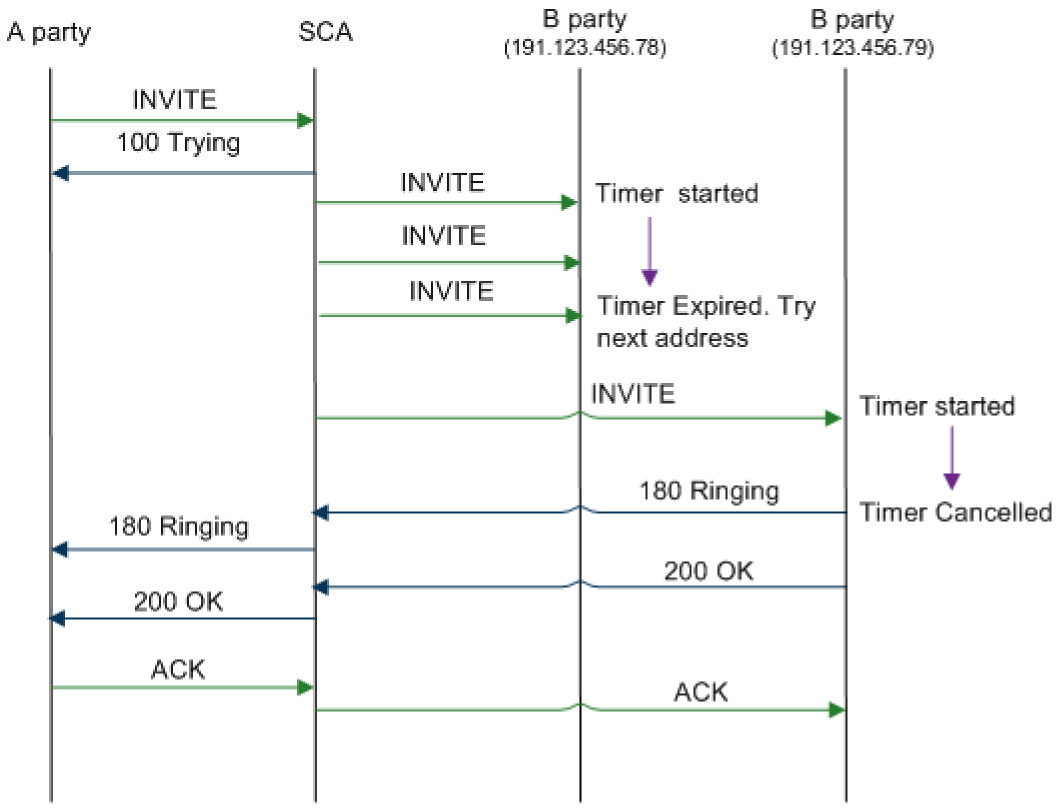
If all the URLs fail the IN will be notified and all active calls to the remote SIP entity will be released. A "Request Timeout" error message is sent to the A-party, unless the A-party addresses have failed or the SCA is directed otherwise by the Control Plan.

The failed address is cached for a configurable amount of time to prevent all subsequent messages being sent to addresses that are known to have failed. A failed address is marked as available when the cache timer expires. If all addresses for a client are detected as failed, this information is not cached and they will all be marked as available.

Note: If the load-balancing behavior is disabled, subsequent messages after a failure is detected are sent to the first address known to be available. If the load-balancing behavior is enabled, the messages are sent to each address in the list using the round robin approach, apart from the addresses marked as DOWN.

Example

Here's an example of a successful call transmission to the B-party's alternate address.



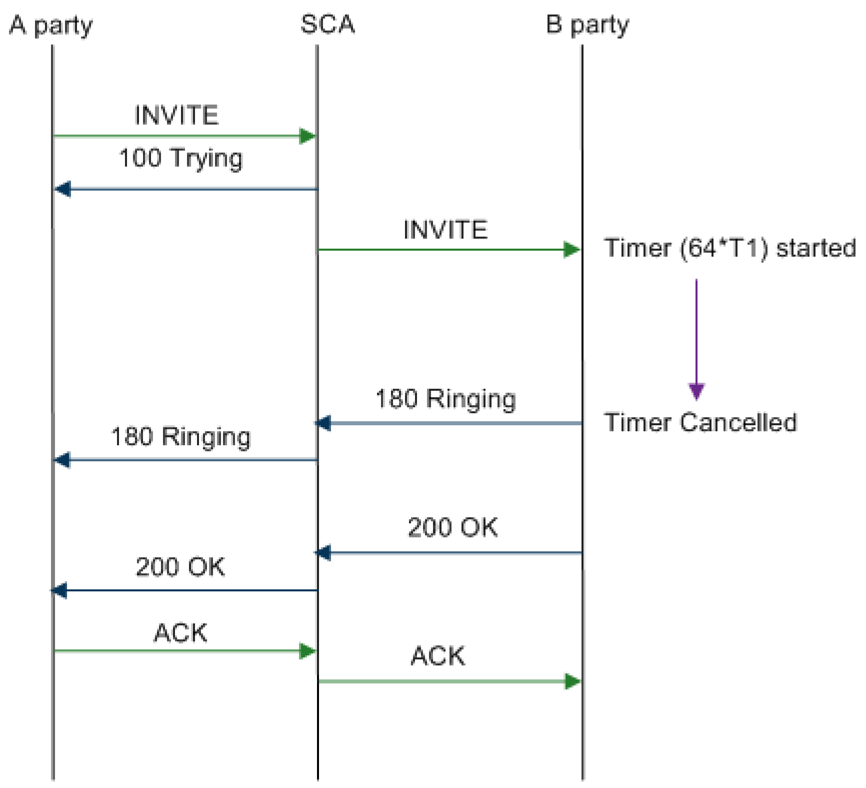
Transaction timer

A transaction timer starts when the SCA sends a request to the B-party. The message is periodically retransmitted after an interval of for 64*T1 seconds, until an appropriate response is received from the other end.

Successful call response

The timer will be canceled when the SCA receives a response from the B-party in the range of 100 to 699 for an INVITE transaction, or a 200 to 699 response for any other request. The default value of T1 is 0.5 seconds, though it can be configured to a different value.

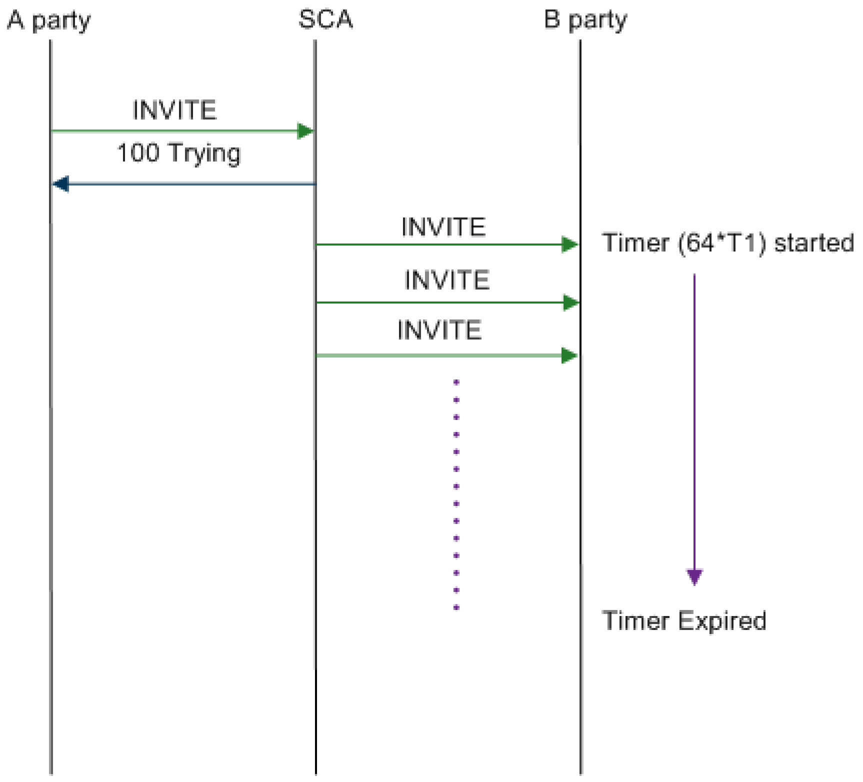
Here's an example of a successful call response received from the B-party.



Remote address failure

If the transaction timer expires and no response is received from the B-party, the SCA notifies the IN and returns a "Request Timeout" error to the A-party, unless directed otherwise by the control plan. This expiry of the transaction timer is interpreted as a remote address failure.

Here is an example of a remote address failure caused due to no response received from the B-party.



Normalization and Denormalization

Introduction

Normalization and denormalization allow for incoming and outgoing numbers to be selected by their prefix and then have numbers stripped or added (as prefix) if necessary. Normalization can be applied to all numbers in an incoming IDP request. Denormalization can be applied to all numbers returned from the IN (Intelligent Network).

Normalization

The SCA performs normalization by modifying the incoming and outgoing numbers to perform prefixing and digit stripping. Normalization is applied to numbers in an IDP message sent to the IN after the next-hop processing.

The following numbers are normalized:

- Called and calling party numbers
- Additional calling party numbers
- Additional called party numbers
- Redirection numbers

Denormalization

The SCA denormalizes numbers when they are returned from the IN. Denormalization is applied to the numbers prior to applying the next hop processing rules.

The SIP protocol does not have the ability to convey nature of address (NoA) information, so the SCA encodes this information using a "+" prefix. If a number returned from the IN has a NoA value of 4, the corresponding number in the outgoing INVITE message is optionally prefixed with a "+" by the SCA. If an incoming INVITE message includes a number prefixed with a "+", then the NoA of the corresponding number sent to the IN will have a NoA of 4. Otherwise a configurable default value will be set.

CLIP and CLIR

Introduction

The SCA performs calling line identification presentation and restriction (CLIP/R) when notified by the incoming message from the A-party.

This CLI information is passed to the SCA in one of the following methods:

- Using the Privacy header in an INVITE message
- Setting the name in the FROM header to “Anonymous”
- Setting the URL in the FROM header to “anonymous@anonymous.invalid”

Privacy header

When the SCA is notified that the A-party CLI information is to be restricted, it will set the Presentation Indicator field of the calling party number to 01 (Presentation Restricted) in the corresponding IDP sent to the IN.

The SCA also ensures that if the Presentation Indicator field of the calling party number is set to 01 by the IN, the outgoing INVITE message includes a Privacy header with a value of “id”.

Setting name in FROM header

If the Display Name parameter in the FROM header of an incoming INVITE message is set to “Anonymous” and the URL in the From header contains an E.164 number, the SCA sets the Presentation Indicator field of the CgPN to 01.

Setting URL in FROM header

If the URL parameter in the FROM header of an incoming INVITE message is set to “anonymous@anonymous.invalid”, the corresponding INVITE message sent by the SCA includes a Privacy header with a value of “id”.

Feature Nodes

ACS feature nodes

This table lists the ACS feature nodes available in the Control Plan Editor, which may be required when using the SCA in conjunction with ACS.

Node name	Node description
Set Presence	This node must be included in the ACS control plan if the SCA will be handling presence and availability requests. It sends a request to a remote presence server to set the availability of a subscriber at a selected location.
Presence Branching	This node must be included in the ACS control plan if the SCA will be handling presence and availability requests.

Node name	Node description
	It sends a request to a remote presence server to check a selected location for the presence and availability of a subscriber.
Copy Hunting List	This node prompts the user for a hunting list and copies the hunting list contents to the Pending TN and Additional Pending TNs buffers.

Tip: For information about Convergent Charging Controller feature nodes, see *Feature Nodes Reference Guide*. For more information about the Control Plan Editor, see *Control Plan Editor User's Guide*.

Chapter 2

Prepaid Platform and NGN Integration

Overview

Introduction

This chapter provides a high-level overview of the integration of IN prepaid systems into next generation networks (NGN). It explains the role of the SCA as an interface between the prepaid platform and the NGN components.

In this chapter

This chapter contains the following topics.

Key Functionality 21
Prepaid Platform Heartbeating 22
Media Description Retrieval from SDP 30
Call ID retrieval 31

Key Functionality

Introduction

The SCA enables real-time charging, instant messaging and personal mobility in SIP-based next generation networks and in the IP multimedia subsystem (3GPP IMS, 3GPP2 MMD). It can be integrated into Oracle's Prepaid Charging solutions based on Charging Control Services (CCS), with on-line charging interfaces to the Convergent Charging Controller Voucher and Wallet Server and to third party real-time billing systems.

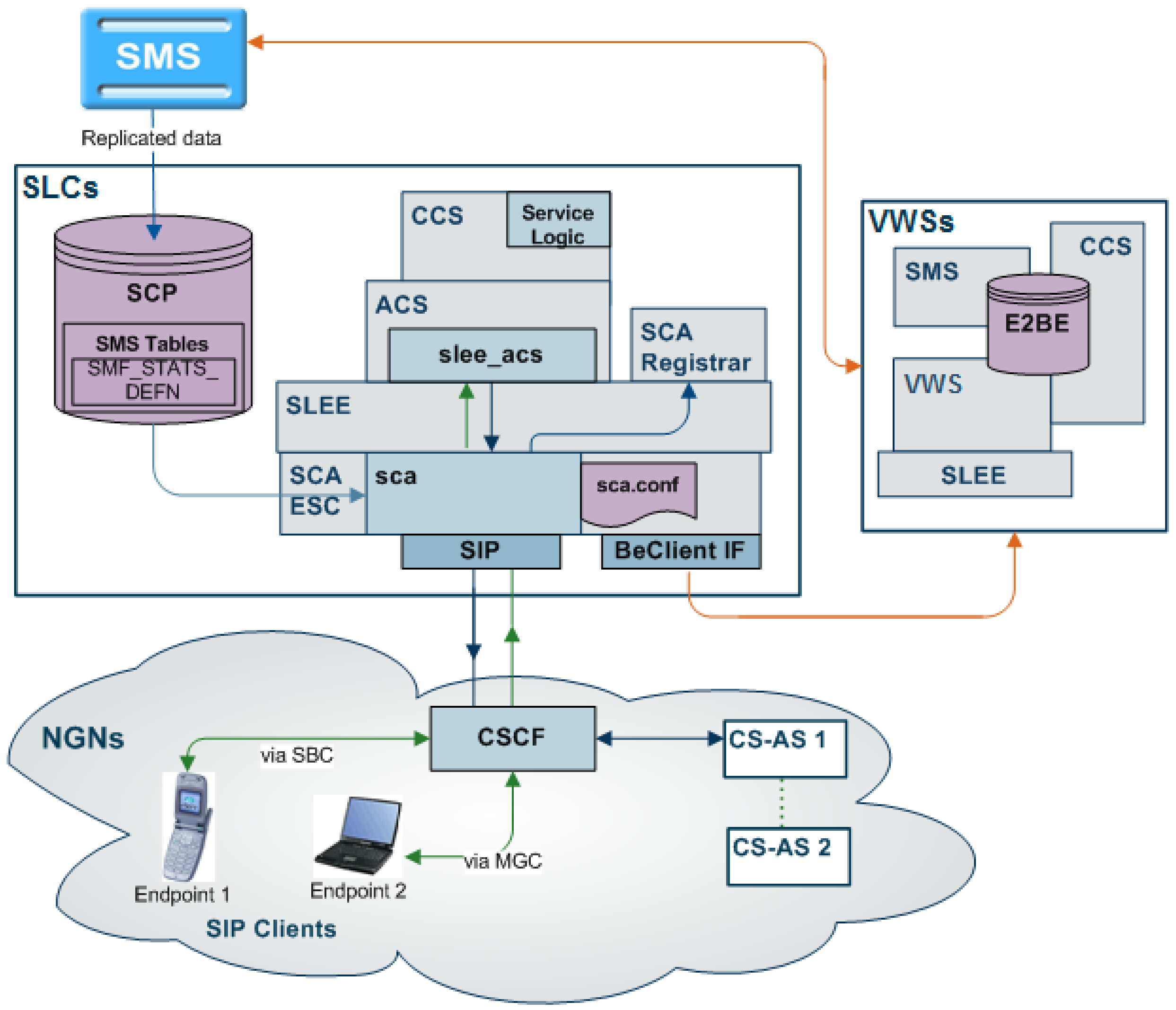
The key features of the SCA that support this integration are:

- Prepaid platform heartbeating towards CSCF
- Media description retrieval from SDP header
- Call ID retrieval
- Status determination of SCA (through OPTIONS)

All messaging between the SCA and the NGN passes through the existing call session control function (CSCF) within the NGN environment.

Diagram

Here is a diagram showing how the Oracle prepaid platform uses the SCA to interface with a NGN.



Prepaid Platform Heartbeating

Introduction

A heartbeating mechanism is used to send periodic SIP OPTIONS messages from the Convergent Charging Controller prepaid IN platform to the CSCF to determine the connectivity status between the two systems. This enables the prepaid platform to cease charging and close down SIP sessions for end users if they are no longer controllable by the IN platform, for example, because a signaling failure has occurred in the network.

SIP OPTIONS heartbeat message

A SIP OPTIONS message is sent to the CSCF at regular time intervals configured within the SCA configuration file. Hence, it is also known as the SIP OPTIONS "heartbeat" message. It is sent to the CSCF regardless of any SIP sessions currently running on the IN platform.

The SCA expects an OK message in response to the OPTIONS heartbeat message. If a response is received, a timer is started and subsequent OPTIONS messages are sent to CSCF at regular intervals. This time interval is configurable in seconds within the SCA configuration file, with a setting of 0 or no entry indicating that no heartbeat messages will be sent.

If a response is not received to an OPTIONS message within the specified time interval, an OPTIONS message is repeatedly sent one of the following:

- An OK message is received
- 64 OPTIONS messages have been retransmitted without an OK message being received

Note: The OPTIONS message is retransmitted at an interval that starts at T1 seconds and doubles until it reaches T2 seconds. It is then retransmitted at an interval of T2 seconds until a total time of 64*T1 seconds has elapsed. T1 and T2 are configurable and default to 0.5 seconds and 4 seconds respectively.

Message header

The message header for SIP OPTIONS heartbeat messages contains the following mandatory fields configurable within the SCA configuration file:

- "To" field
- "From" field
- Request-URI
- Route
- P-Asserted-Identity (including username)
- P-Charging-Vector (optional)

Example

Here is an example of a OPTIONS heartbeat message sent from the prepaid platform to CSCF:

```
OPTIONS sip:bsas.sip.teleco.pl;transport=UDP; SIP/2.0
Via: SIP/2.0/UDP 12.0.0.21:6060
Max-Forwards: 70
To: <sip:bsas.sip.teleco.pl;transport=UDP>
From: <sip:prepaid.sip.teleco.pl;transport=UDP>;tag=1234567890
Call-ID: prepaid1234@12.3.4.56
CSeq: 10 OPTIONS
Route: <sip:scscf.sip.teleco.pl:6060;lr>
Contact: <sip:12.0.0.21:6060;transport=UDP>
Accept: application/sdp
P-Asserted-Identity: "45678901234" <sip:45678901234@sip.teleco.pl>
Content-Length: 0
P-Charging-Vector:
icid-value=prepaid34567aae0;orig-ioi=prepaid.sip.teleco.pl
```

Prepaid Heartbeat Process

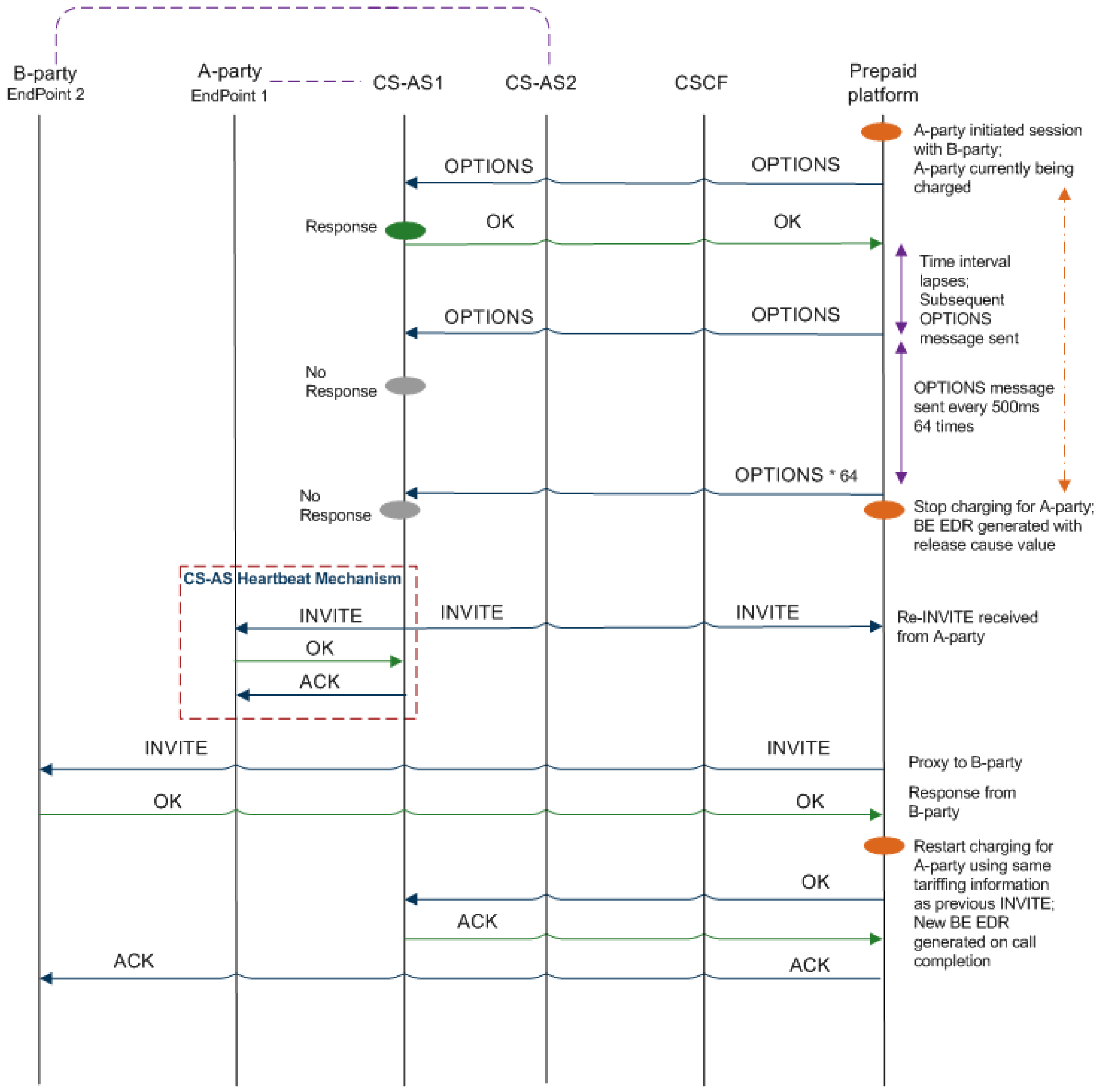
The detailed process is explained below:

Step	Action
1	The prepaid IN platform sends a SIP OPTIONS heartbeat message to the CSCF and awaits an OK message.
2	If a response:

Step	Action
	• Is received, a timer is started to send subsequent OPTIONS messages to CSCF at regular intervals. • Is *not* received within a specified period, an OPTIONS message is retransmitted until an OK message is received or 64 OPTIONS messages have been retransmitted without any response. **Note:** The OPTIONS message is retransmitted at an interval that starts at T1 seconds and doubles until it reaches T2 seconds. It is then retransmitted at an interval of T2 seconds until a total time of 64*T1 seconds has elapsed. T1 and T2 are configurable and default to 0.5 seconds and 4 seconds respectively.
3	A critical alarm is raised and charging for current SIP sessions is stopped, that is, all concerned SIP sessions are placed in a “Charging void” state.
4	A VWS EDR is generated with an appropriate release cause value as configured in the SCA configuration file.
5	The SCA notifies the IN to release the call using a RELEASE message that indicates the release cause value.
6	If the restart charging option is enabled, the SCA will attempt to restart the charging session upon receiving a re-INVITE message from the originating party, which is successfully proxied to responded by the B-party. For details about the procedure, see *Restarting charging session* (on page 28).

Diagram

Here is an illustration of the prepaid platform heartbeating mechanism. The resulting CS-AS heartbeating activity is highlighted in red.


B-party EndPoint 2
A-party EndPoint 1
CS-AS1
CS-AS2
CSCF
Prepaid platform
A-party initiated session with B-party; A-party currently being charged
OPTIONS
OPTIONS
Response
OK
OK
Time interval lapses; Subsequent OPTIONS message sent
OPTIONS
OPTIONS
No Response
OPTIONS message sent every 500ms 64 times
OPTIONS * 64
No Response
Stop charging for A-party; BE EDR generated with release cause value
CS-AS Heartbeat Mechanism
INVITE
INVITE
INVITE
Re-INVITE received from A-party
OK
ACK
INVITE
INVITE
Proxy to B-party
OK
OK
Response from B-party
Restart charging for A-party using same tariffing information as previous INVITE; New BE EDR generated on call completion
OK
ACK
ACK
ACK

Stopping charging sessions

The prepaid heartbeating mechanism ensures that charging can be stopped for SIP endpoints affected by a signaling problem between the prepaid platform and CSCF.

When 64 OPTIONS messages are retransmitted without any response, a critical alarm is raised as per the alarm details configured within the SCA configuration file. Charging for current SIP sessions is stopped and all concerned SIP sessions are placed in a 'charging void' state.

A VWS EDR is generated with the appropriate release cause value. For more details, see *release cause values* (on page 26).

Release cause values

When the charging is closed for a SIP session, a VWS EDR is generated with the appropriate release cause value.

This value is configurable within the SCA configuration file, the permissible range being 0 to 127, with a default value of 55. The release cause value written to the VWS EDR specified for a CS-AS heartbeat mechanism failure is also configurable within the SCA configuration file, the permissible range being 0 to 127, with a default value of 60.

The SCA uses this value in a RELEASE message to disconnect the call to the IN service. The IN service then includes the value in the VWS EDR, allowing a user's account to be credited appropriately if the release cause value is detected during EDR post-processing. It also alerts the EDR post-processing system if more than one EDR exist for a call (in case a new EDR is generated when charging restarts).

Warning: If a release cause value is specified outside the permissible range an appropriate alarm is raised and the SCA will fail to restart.

Release cause mapping

This table lists valid INAP release cause codes, and the SIP error codes to which they are mapped.

Code	INAP Release Cause	Code	SIP Error Code
1	Unallocated Number	404	Not Found
2	No Route To Network	404	Not Found
3	No Route To Destination	404	Not Found
17	User Busy	486	Busy Here
18	No User Responding	408	Request Timeout
19	No Answer From User	480	Temporarily Unavailable
20	Subscriber Absent	480	Temporarily Unavailable
21	Call Rejected	603	Decline
22	Number Changed	410	Gone
23	Redirection To New Dest	410	Gone
26	Non-Selected User Clearing	404	Not Found
27	Destination Out of Order	502	Bad Gateway
28	Address Incomplete	484	Address Incomplete
29	Facility Rejected	501	Not Implemented
31	Normal	404	Not Found
34	No Circuit Available	503	Service Unavailable
38	Network Out of Order	503	Service Unavailable
41	Temporary Failure	503	Service Unavailable
42	Switching Equipment Congestion	503	Service Unavailable
47	Resource Unavailable	503	Service Unavailable
55	Incoming calls barred within CUG	403	Forbidden
57	Bearer capability not authorized	503	Service Unavailable
58	Bearer capability not presently available	503	Service Unavailable
65	Bearer capability not implemented	488	Not Acceptable Here
70	Only restricted digital available	488	Not Acceptable Here
79	Service/option not implemented	501	Not implemented

Code	INAP Release Cause	Code	SIP Error Code
87	User not member of CUG	403	Forbidden
88	Incompatible destination	503	Service unavailable
102	Recovery on timer expiry	504	Gateway timeout
111	Protocol error	500	Server internal error
127	Interworking unspecified	500	Server internal error

Note: These mappings can be ignored by setting the `propagate_b_error = true` configuration item.

This will always ignore the above mapping and just forward the error response from the B-leg. Beware that this will mean that the IN release cause will always be ignored.

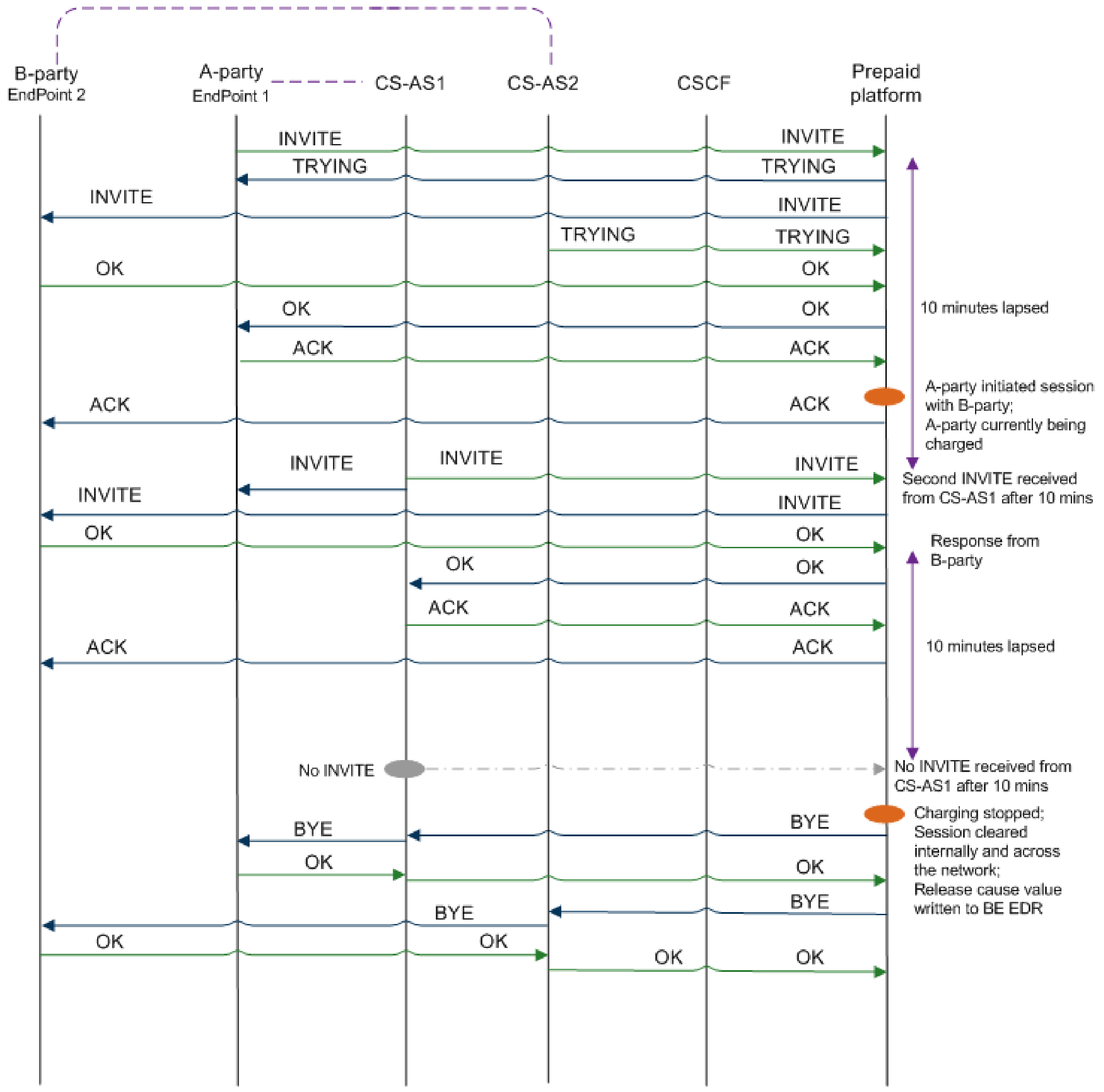
CS-AS heartbeating mechanism

A heartbeat message is sent to the prepaid IN platform by the CS-AS (application server) associated with each endpoint to indicate that the SCA must be able to handle all endpoints currently involved within the SIP session. This is known as CS-AS heartbeat mechanism.

This allows the prepaid IN platform to determine signaling problems within the network. If a heartbeat message is not received within the specified time limit, the prepaid IN platform will end the SIP session and cease charging for the associated SIP endpoint.

Diagram

Here is an illustration of the CS-AS heartbeating mechanism.


B-party
EndPoint 2
A-party
EndPoint 1
CS-AS1
CS-AS2
CSCF
Prepaid platform
INVITE
TRYING
INVITE
TRYING
OK
ACK
10 minutes lapsed
A-party initiated session with B-party; A-party currently being charged
Second INVITE received from CS-AS1 after 10 mins
Response from B-party
10 minutes lapsed
No INVITE
No INVITE received from CS-AS1 after 10 mins
Charging stopped; Session cleared internally and across the network; Release cause value written to BE EDR
BYE
OK

Restarting charging sessions

The CS-AS heartbeat mechanism enables restarting charging sessions that have entered the Charging void state by periodically checking the connectivity status of the endpoints.

This process is initiated if a re-INVITE message received from the A-party (originating endpoint connected to a CS-AS), proxied by the SCA to the B-party (destination endpoint) results in a successful OK message from the B-party.

When an attempt is made to resume a charging session, the SCA will add a configurable prefix to the called party address sent to the charging system so that the IN service can identify the resumed call and suppress IVR messages.

Note: The SCA will attempt to resume a session in the Charging void state on if the resume charging option is enabled in the SCA configuration file.

Process

The detailed restart charging process is explained below.

Stage	Description
1	The CS-AS sends a periodic INVITE heartbeat message to the SCA on behalf of each endpoint involved in a SIP session.
2	The SCA receives the message from the A-party, proxies it to the B-party (destination endpoint) and awaits an OK response.
3	When the B-party responds with an OK, the SCA will recommence charging for A-party if the restart charging option is enabled within the SCA configuration file.
4	The contents of the new INVITE are used to determine charging parameters, however, if there is insufficient information, the charging parameters of the previous INVITE will be used.
5	If the attempt to restart charging succeeds, then the OK message received from the B-party is sent to the A-party and a re-INVITE timer is restarted to receive subsequent heartbeat messages.
6	If the attempt to restart charging fails, then the session will be cleared by sending BYE messages to both endpoints.

Note: The SCA interprets an INVITE message as a heartbeat message if it has the same call ID value in the message header as an outstanding SIP session.

SCA message processing

The following scenarios describe the SCA action's response to possible messages received from the CSCF when a SIP session is placed in the Charging void state.

Scenario	Action
No heartbeat INVITE message or non-terminating SIP message	If no periodic INVITE message or non-terminating SIP message is received within the configured time, the SIP session is closed and charging is stopped.
Re-INVITE message received from the A-party	The SCA will proxy it to B-party and await OK message.
OK message is received from the B-party	The SCA will attempt to resume the charging session as per the configuration options set in the SCA configuration file. Charging information, if present in new INVITE is used, else contents of previous message are used.
Re-INVITE received for ongoing call currently being charged	The SCA will proxy and then discarded this message.
Re-INVITE message received from B-party	The SCA will proxy the message to A-party and take no further action.
Any non-terminating message other than INVITE received A-party	The SCA will proxy the message to the B-party; the re-Invite timer is restarted when B-party responds with an OK.

Scenario	Action
Any non-terminating message other than INVITE received from A-party for sessions in Charging void state	The SCA will attempt to resume the charging session as per the configuration options set in the SCA configuration file. The message is proxied to the B-party and charging commences when an OK is received from B-party. **Notes:** Charging parameters will be used from previous charging parameters unless an UPDATE is received overriding previous message information.

Media Description Retrieval from SDP

Introduction

The SIP endpoints use the SDP portion of the initial INVITE message to indicate the type of media session that should be initiated. This information is used by the SCA to pass tariff information to the charging components.

The tariff can be re-evaluated by the prepaid platform during session setup or progress. The session setup stage includes media session negotiation. During a call, the media type and tariff can be updated using a re-INVITE or UPDATE message. The final agreed media type will be the type for which the call is billed.

This media type information is written to the EDR generated for the session, by using the Branch on Bearer Type node in combination with the Set BE EDR node. For more information on feature nodes, refer to *Feature Nodes Reference Guide*.

Media description field

The media description field in the message body of the initial INVITE received for a session is used to derive the tariff applied to the whole session.

The SCA searches for more than one occurrence of the Media description field and its associated direction media attributes as more than one media can be applied to a call.

An example is shown below for a session with both audio and video:

```
Media Description, name and address (m): audio 21358 RTP/AVP 107 119 0
Media Attribute (a): sendrecv
Media Description, name and address (m): video 21360 RTP/AVP 115 34
Media Attribute (a): sendrecv
```

Media description string

The SCA looks for the occurrence of the following media description string within the message body:

```
Media Description, name and address (m):
```

The strings of interest generally associated with this tag are “audio” and “video”.

Example: An entry is shown below for an audio call:

```
Media Description, name and address (m): audio 12345 RTP/AVP 100 100 0
```

Media description attribute

After an occurrence of the media description string is found, the SCA looks for an occurrence of the media attribute field that indicates if the type of media is send, send or receive.

Example: An example entry is shown below for a bi-directional media stream:

```
    Media Attribute (a): sendrecv
```

Media mapping table

Once the SCA has completed deciphering the media description/attribute parameters from the INVITE it looks up a media mapping table that indicates the bearer capability (tariff) to be used for the session.

The media mapping table is configured within the SCA configuration file and allows various combinations of media description/attributes to be mapped to a bearer capability integer value. Unique values of bearer capabilities must be used so that the control plan executed by the IN service can check for a particular bearer capability and write the correct media information to the EDR that is generated.

The result of not finding a bearer capability mapping in the media mapping table can be configured in the SCA configuration file as:

- An alarm is raised showing the media description/attributes that have no mapping. The session start-up is terminated and an appropriate error message is sent to the initiating party.
- The bearer capability is set to a default value in the SCA configuration file (blank setting indicates that field is not populated) and the call continues.

Note: If an UPDATE message is received in a call not in the Charging void state is either rejected, ignored or proxied as configured in the SCA configuration file. The SDP information cannot be used to update charging information.

Example

Here are a few sample entries for a media mapping table:

Media Attribute	Description	Bearer Capability
audio:sendrecv::1	Indicates for only Audio - sendrecv	1
video:sendrecv::2	Indicates for only Video - sendrecv	2
video:send::3	Indicates for only Video send	3
video:recv::4	Indicates for only Video receive	4
audio:sendrecv,video:sendrecv::5	Indicates for Audio sendrecv and Video send/recv	5
audio:sendrecv,video:send::6	Indicates for Audio sendrecv and Video send	6
audio:sendrecv,video: recv::7	Indicates for Audio sendrecv and Video recv	7
audio:inactive,video: recv::8	Indicates for Audio inactive and Video recv	8
audio:inactive,video: inactive::9	Indicates for Audio inactive and Video inactive	9

Call ID retrieval

Introduction

The Call ID in the initial INVITE received while starting a SIP session must be included in all EDRs generated by the prepaid platform for the SIP session. This helps correlate the EDRs generated for that session by other applications across various platforms.

The Call ID is written to the VWS EDR when the Set BE EDR node is triggered in a call Control Plan. However, to write the Call ID in ASCII form to the EDR an MSC Address must be configured in the SCA configuration file and a matching MSC address must be entered in the **eserv.config** file and set up for ASCII decoding.

Call ID field

The Call ID field in the message header of the initial SIP INVITE received from the A-party is passed to the call control plan. It is then written to the VWS EDRs by the Set BE EDR feature node.

The Call ID field is placed within the CAMEL call reference field within the IDP sent by the SCA to trigger ACS. The maximum length of this field is 64 characters. If the Call ID is greater than 64 characters, then the first 64 characters will be displayed in the EDR field and an alarm will be raised to indicate that the Call ID has been truncated. The details of this alarm can be configured in the SCA configuration file to display:

- Contents of 'To' field
- Contents of 'From' field
- Truncated Call ID
- Actual Call ID

Also, the SCA can be configured to operate in a back-to-back user agent (B2BUA) mode, so that the call ID sent to the B-party is generated by the IN platform with a recognizable prefix.

Chapter 3

Scenarios

Overview

Introduction

This chapter explains how common scenarios are handled by the Session Control Agent.

In this chapter

This chapter contains the following topics.

Call Forwarding 33
Call Redirection 36
Callback 36
SIP Error Response 40

Call Forwarding

Introduction

The most common use of the SCA is to trigger a control plan to process calls in conjunction with the IN call model. This is referred to as an “IN controlled call”. Using this method the SCA acts as a translator between the SIP protocol used in the VOIP network and the TCAP/INAP protocols used to communicate with the control plan.

This example scenario shows how the SCA (running in B2BUA mode) can be used to forward calls. In this scenario a SIP request triggers a control plan which changes the call destination number and forwards the request accordingly.

Call Forwarding Process

The call forwarding process is described below.

Stage	Description
1	A SIP INVITE request is received on UDP port 5060.

Example INVITE request:

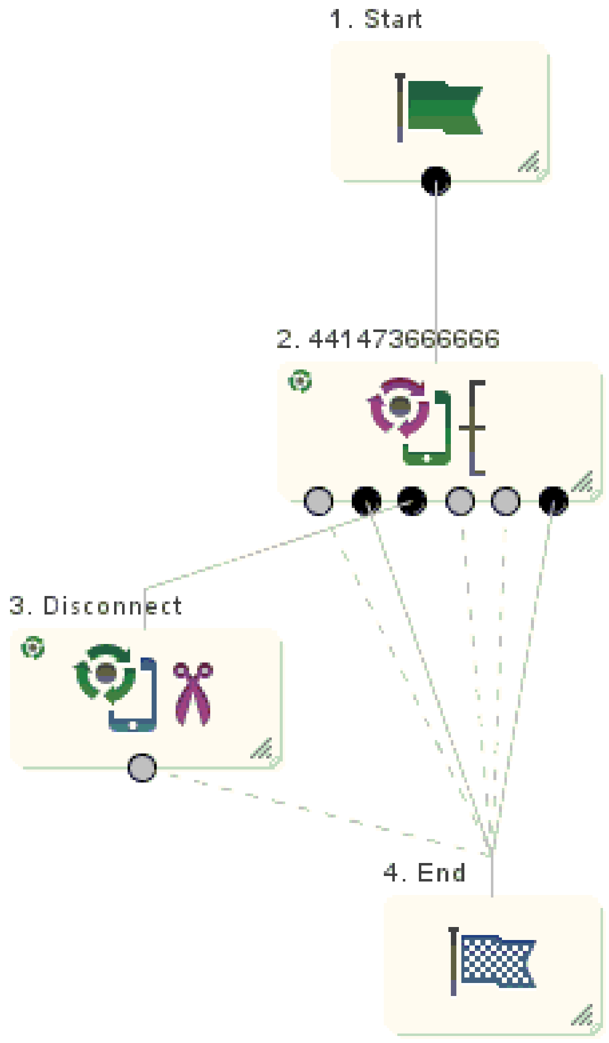
```
INVITE sip:441473289900@oracle.com SIP/2.0
Via: SIP/2.0/UDP 192.168.25.80:5060;branch=z9hG4bKaaa
From: 441473555555<sip:441473555555@192.168.25.80>;tag=1
To: 441473289900 <sip:441473289900@oracle.com>
Call-ID: 1-8959@192.168.25.80
CSeq: 1 INVITE
Contact: sip:441473555555@192.168.25.80:5060
Max-Forwards: 70
Content-Type: application/sdp
Content-Length: 124
v=0
o=user1 53655765 2353687637 IN IP4 192.168.25.80
s=-
c=IN IP4 192.168.25.80
t=0 0
m=audio 21358 RTP/AVP 107 119 0
```

Stage	Description
2	The SCA queries the **rules.nhp** configuration file and looks for a rule matching the INVITE message. In this example the INVITE message triggers the "acs" default rule.

Example Rules.nhp:

```
RULES = {
  "acs" = ( DEFAULT ) IDP ( SK = 110 );
  "fwd" = ( TO USER MATCHES "666666" )
                  FORWARD ( URI="sip:192.168.25.82");
}
HOST * = {
   METHOD "INVITE" = { "acs", "fwd" }
}
```

Stage	Description
3	The rule translates the INVITE message into an IDP message. For details on how the message is translated, see *Invite message translation* (on page 35).
4	The IDP message is forwarded to ACS on service key 110.
5	The IDP triggers a simple control plan containing an Attempt Termination node. **Example:**



Stage	Description
6	The Attempt Termination node redirects the number to 441473666666. A Connect message is sent back to the SCA with this as the new called party number.

Stage	Description

7	The SCA generates a new INVITE request for the second part of the call. The request contains the new CdPN in the To header and the received CgPN in the From header.
	Example INVITE request:
	`INVITE sip:441473666666@oracle.com SIP/2.0`
	`Via: SIP/2.0/UDP 192.168.26.182:5060;branch=z9hG4bK946513216548`
	`From:441473555555 <sip:441473555555@oracle.com>;tag=2895685`
	`To: 441473666666 <sip:441473666666@oracle.com>`
	`Call-ID: ESGSCA12345689746541321@192.168.26.182`
	`CSeq: 1 INVITE`
	`Contact: sip:441473555555@192.168.26.182:5060`
	`Max-Forwards: 70`
	`Content-Type: application/sdp`
	`Content-Length: 124`
	`v=0`
	`o=user1 53655765 2353687637 IN IP4 192.168.25.80`
	`s=-`
	`c=IN IP4 192.168.25.80`
	`t=0 0`
	`m=audio 21358 RTP/AVP 107 119 0`

8	ACS returns E.164 numbers which must be translated to a SIP URI. For details, see *E.164 number translation* (on page 35).
9	The "fwd" rule in **rules.nhp** (the rule matching the username part of the new "To" header) is fired. This forwards the request to the specified destination. In this case: 192.168.25.82.
10	A 200-OK success response is received, indicating that the B-party has answered.
11	A release cause of 31 is sent to the control plan which takes the success branch exit.

Invite message translation

The SCA uses the following process to translate between a SIP INVITE message and an INAP IDP.

1. The username part (before the "@") of the SIP URI in the From header is used as the CallingParty number (for example, 441473555555). If the number is preceded by a "+" character, then the NoA of the CgPN is set to 4. Otherwise it is set to a configurable default value.
2. The username part of the To header URI becomes the Called Party number (CdPN).
3. If present, the URI in the topmost Diversion header becomes the Redirecting Party ID.
4. If present, the P-Asserted-ID header can optionally be used as the CgPN.
5. Other default values can be configured in the **sca.config** file and **tdp.conf** files. For example, the **tdp.conf** file sets information such as the Switch type for the trigger detection point information.

Note: For more information on the SCA configuration files, refer to *Configuration* (on page 41).

E.164 number translation

An E.164 number is translated to a SIP URL by adding a domain name to the end of the number. For example, this method could be used to translate the To header:

```
To: {CdPN} <sip:{CdPN}@{default_domain}>
```

Where:

The default_domain configuration option contains a URI, such as, oracle.com and this is appended to the E.164 Called-party number to create a SIP URI.

Note:

- The From URI has been changed (the default_domain is used).

- The trans_mapping configuration option allows different domains to be used depending on prefix.
- The destination address in the To header is the same as the Request-URI (the first line of the SIP message).
- The displayname part of the To and From headers have been changed. If required, the original displayname can be retained using the `trans_behaviour` configuration option.
- The To URI and Request-URI can be overridden with the URI in the "fwd" rule by specifying the SET-URI configuration option.

Call Redirection

Introduction

Call redirection is a simple scenario often used for ported number queries.

Example redirection rules

Here is an example of call redirection configuration in the **rules.nhp** file.

```
RULES = {
  "redirect" = ( DEFAULT ) REDIRECT ( SK = 110 );
}
HOST * = {
   METHOD "INVITE" = { "redirect" }
}
```

Call Redirection Process

The call redirection process is described below.

Stage	Action
1	An INVITE request triggers ACS on service key 110.
2	The called party number returned from ACS is translated to a SIP URI and sent to the A-party in the contact header of a "302 Moved Temporarily" response. For more information on number translation, see *E.164 number translation* (on page 35).
3	The A-party re-sends the INVITE request to the URI specified in this contact header.

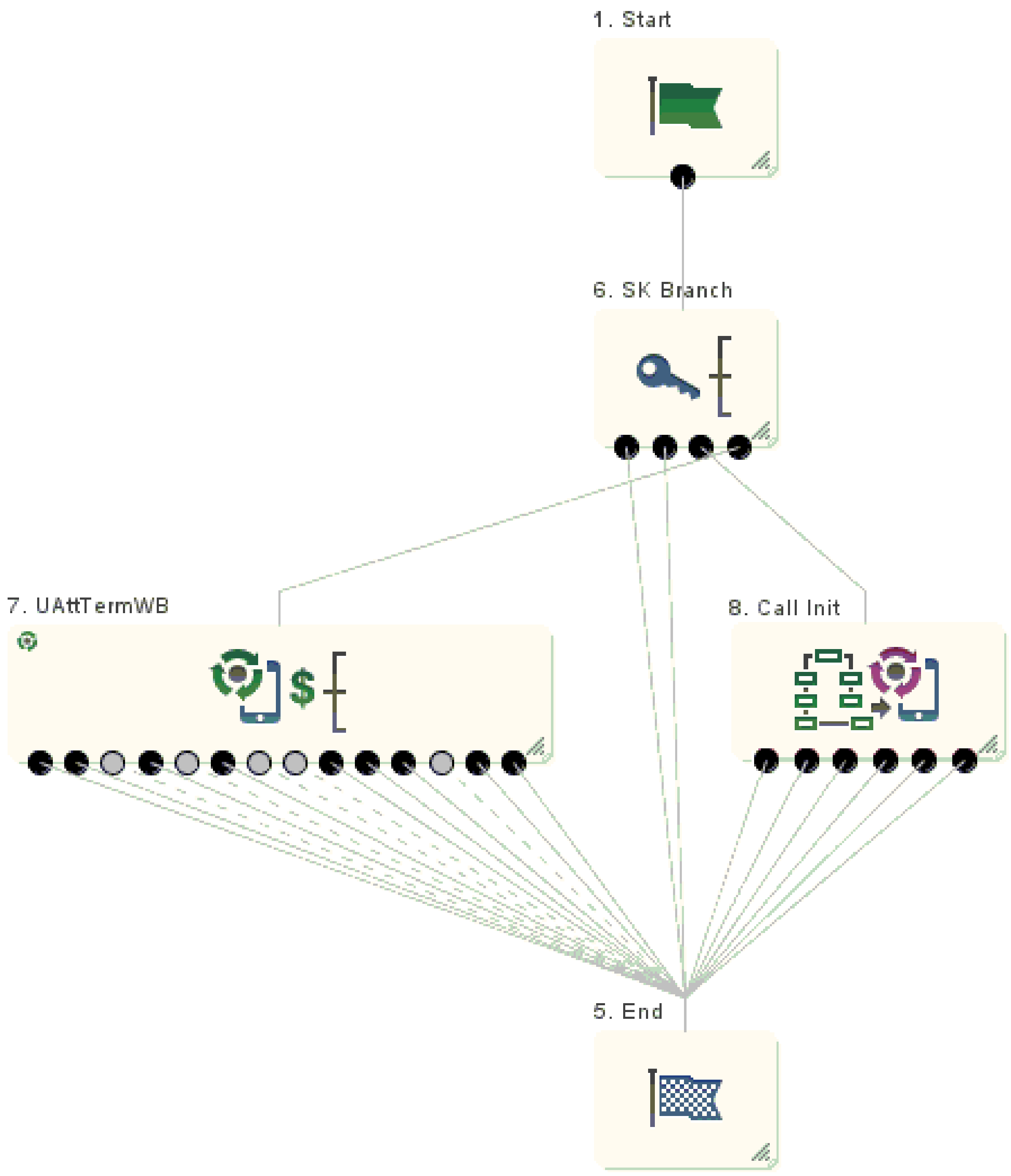
Callback

Introduction

Callback or Third Party Call Control allows ACS to initiate calls between two parties.

Callback control plan diagram

Here is an example control plan for handling Callback.



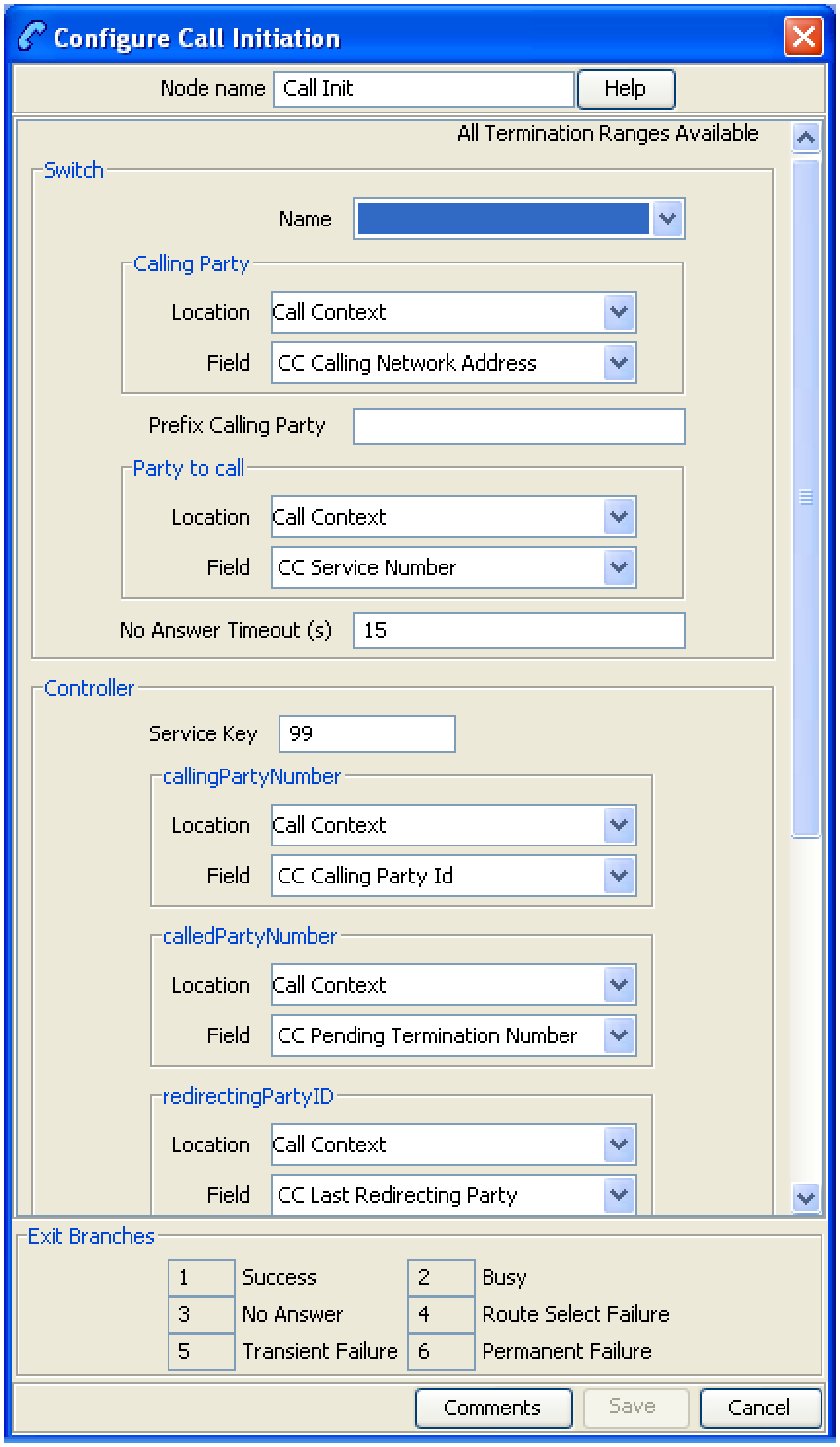
Callback Process

This process explains how Callback works.

Stage	Description
1	The first leg of the call is triggered by the Call Initiation node. The Switch and Calling Party sections configured in the node, specify the destination of the initial INVITE request.
2	The INVITE message is created with a To header containing the "Party To Call" and a From header containing the "Calling Party".
3	The rule from the **rules.nhp** file matching the INVITE message is triggered.
4	When the A-party answers the call, the Call Initiation node transfers the dialog to a different control plan by sending an IDP to ACS (as if the call had been initiated by the A-party).
5	The IDP is sent on the service key configured in the node. It contains the CgPN and CdPN configured in the Controller section of the node.
6	The SCA sends an INVITE to the CdPN, thus connecting the two call legs together.

Configure Call Initiation example

Here is an example Configure Call Initiation screen showing how the node may be configured for callback.


Configure Call Initiation

Node name: Call Init

All Termination Ranges Available

Switch

Name:

Calling Party

Location: Call Context

Field: CC Calling Network Address

Prefix Calling Party:

Party to call

Location: Call Context

Field: CC Service Number

No Answer Timeout (s): 15

Controller

Service Key: 99

callingPartyNumber

Location: Call Context

Field: CC Calling Party Id

calledPartyNumber

Location: Call Context

Field: CC Pending Termination Number

redirectingPartyID

Location: Call Context

Field: CC Last Redirecting Party

Exit Branches

1	Success	2	Busy
3	No Answer	4	Route Select Failure
5	Transient Failure	6	Permanent Failure

SIP Error Response

Introduction

This scenario explains how the SCA handles SIP error responses.

SIP error response example

This example shows a SIP 404 error response to a call forward request.

```
SIP/2.0 404 Not Found
Via: SIP/2.0/UDP 192.168.26.182:5060;branch=z9hG4bK946513216548
From:441473555555 <sip:441473555555@oracle.com>;tag=2895685
To: 441473666666 <sip:441473666666@oracle.com>;tag=12135612
Call-ID: ESGSCA12345689746541321@192.168.26.182
CSeq: 1 INVITE
Reason: Q.850;cause=1
Content-Length: 0
```

SIP error response process

The detailed process is described below.

Stage	Description
1	The SIP error response code is mapped to a release cause and sent to ACS. In this example the error code 404 and error cause 'Not Found' is mapped to the release cause 1 and sent to ACS. For details of SIP error response mappings, refer to *Release cause mapping* (on page 26).
2	The control plan sends the release cause back to the SCA.
3	The SCA performs one of the following: • Maps the received release cause back to a SIP error code. This will always happen if the control plan instructed the call to be released (for example, using a Disconnect node with a release cause on the “no-answer” branch). • Forwards the error it received from the B-leg back to the A-leg, ignoring the release cause from ACS. To do this you must specify the propagate_b_error flag in **sca.config**.

Chapter 4

Configuration

Overview

Introduction

This chapter explains how to configure the Oracle Communications Convergent Charging Controller application.

In this chapter

This chapter contains the following topics.

Configuration Overview 41
sca.config Configuration 42
SCA Configuration 43
ESC Configuration 77
Parser Configuration 84
Presence Configuration 85
acs.conf Configuration 87
SLEE.cfg Configuration 88
Configuring EDR Collection 88
Configuring IN Call Model Triggers 89

Configuration Overview

Introdution

This topic provides a high level overview of how the Session Control Agent (SCA) is configured.

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

Configuration components

The Session Control Agent is configured by the following components:

Component	Locations	Description	Further Information
SCA **sca.config**	All SCA SLC platforms	The SCA is configured by the `sca` section of **sca.config**.	*SCA Configuration* (on page 43).
ESC **sca.config**	All SCA SLC platforms	The Oracle SIP Chassis (ESC) is configured by the `esc` section of **sca.config**.	*ESC Configuration* (on page 77).
eserv.config	All SCA SLC platforms	The translations to SIP availability and location definitions are configured by the `presence` section of **eserv.config**.	*Presence Configuration* (on page 85).

Component	Locations	Description	Further Information
acs.conf	All SCA SLC platforms	The acsChassis plugin library for presence querying and setting is configured by the `acsChassis` section of **acs.conf**.	*acs.conf Configuration* (on page 87) and the *Advanced Control Services Technical Guide.*
SLEE.cfg	All SCA SLC platforms	The SLEE interface is configured to include the SCA interface.	*SLEE.cfg Configuration* (on page 88) and the *Service Logic Execution Environment Technical Guide.*
rules.nhp	All SCA SLC platforms	The SCA Next Hop Processing is configured by the **rules.nhp** file.	*NHP Configuration* (on page 93).
tdp.conf	All SCA SLC platforms	The Trigger Point Definition file for triggering SLEE requests to external IN applications such as ACS.	*Configuring IN Call Model Triggers* (on page 89).
stats_config file	All SCA SLC platforms	Lists the SCA statistics. Required for collecting SCA statistics where ORACLE is not installed.	*Statistics* (on page 108)

sca.config Configuration

Introduction

The **sca.config** file is used to configure the Session Control Agent. It contains different sections defining data relevant to the SCA itself, and the Oracle SIP Chassis (ESC shared library). It is located in the **/IN/service_packages/SCA/etc** directory:

The **sca.config** file format uses hierarchical groupings to divide up the options into logical groups.

Configuration file format

To organize the configuration data within the **sca.config** file, some sections are nested within other sections. Configuration details are opened and closed using either { } or [ ].

- Groups of parameters are enclosed with curly brackets - { }
- An array of parameters is enclosed in square brackets - [ ]
- Comments are prefaced with a # at the beginning of the line

To list things within a group or an array, elements must be separated by at least one comma or at least one line break. Any of the following formats may be used:

```
{ name="route6", id = 3, prefixes = [ "00000148", "0000473"] }
{ name="route7", id = 4, prefixes = [ "000001049" ] }
```

or

```
{  name="route6"
    id = 3
    prefixes = [
        "00000148"
        "0000473"
    ]
}
{  name="route7"
    id = 4
    prefixes = [
            "000001049"
        ]
    }
```

or

```
    {  name="route6"
        id = 3
        prefixes = [ "00000148", "0000473" ]
    }
    {  name="route7", id = 4
        prefixes = [ "000001049" ]
    }
```

Editing the file

Open the **sca.config** file using a standard file editor. Do not use file editors such as Microsoft Word that attach Microsoft DOS or Windows line termination characters (that is, ^M) at the end of each row, as this will cause file errors when the application tries to read the configuration file.

Always keep a backup of your **sca.config** before making any changes to it, to ensure that you always have a working copy.

Loading sca.config configuration changes

If you change the configuration file, then you must send a signal (SIGHUP) to the sca process, or restart the SLEE, to enable the new options to take effect.

SCA Configuration

Introduction

The `sca` section in the **sca.config** file must be configured to enable the SCA to work. An example **sca.config** file showing all the available configuration options is installed by the scaScp package in **/IN/services_packages/SCA/etc/sca.config.all.example**.

The **sca.config** file needs to be present on all SCA SLCs.

Note: All mandatory configuration in **sca.config** is done at installation time by the configuration script.

Example sca configuration

The following is an example of `sca` configuration in **sca.config**:

```
sca = {

    local_ip = "192.168.0.1"

    enableCDRs = true
    cdrTempDir = "/IN/service_packages/SCA/cdr/open"
    cdrFinalDir = "/IN/service_packages/SCA/cdr/closed"
    cdrSizeLimit = 100000
    use_ALegCallID = true
    local_contact = false

    cdrAgeLimit = 600

    rules = "/IN/service_packages/SCA/etc/rules.nhp"

    b2bua = true
```

```
noin_dropcall = false

inap_noa_calling_party = 4
inap_noa_called_party = 4
inap_noa_redirecting_party = 4
inap_scr = 3
inap_pres = 0
inap_numplan = 1
inap_inn = true

usePAssertedID = false

oracleUserAndPassword = "/"

rem_comm_port = 3615
propagate_b_error = false

remotePartyIdTrans = "NEVER"
include_rpi_privacy = false
update_rpi_privacy = false
include_rpi_screen = false
update_rpi_screen = false
include_rpi_pty_type = false
rpi_pty_type = "calling"
include_rpi_id_type = false
rpi_id_type = user
rpi_presentation_allowed = "full"
rpi_presentationRestricted = "off"
rpi_addressNotAvailable = "uri"
rpi_spare = "name";
rpi_user_not_verified = "no"
rpi_user_verified_passed = "yes"
rpi_user_verified_failed = "no"
rpi_network_provided = "no"

p_asserted_identity_trans = "NEVER"
include_pai_tel_header = false
cf_use_cdpn_from_request = false

Registrar = {
    CacheSize = 0
    DefaultExpiry = 3600
}

load_balancing_enabled=true

dns_cache_time=600

failed_address_timeout=300

invite_failover_only=true

replace_diversion_header=false
update_diversion_header=true

### Configuration for heartbeating to a gateway ###

heartbeat_send_interval=30
heartbeat_send_timeout = 10
heartbeat_destination="gateway_rule"
heartbeat_to_address="gateway.uk.oracle.com"
heartbeat_from_address="uas.uk.oracle.com"
heartbeat_request_uri="gateway.uk.oracle.com"
heartbeat_route="gateway.uk.oracle.com"
```

```
heartbeat_p_asserted_id = "\"uk.oracle.com\" <sip:01473289900@sip.uk.oracle.com>"
heartbeat_p_charging_vector = "icid-value=prepaid313264321646132;orig-
ioi=prepaid.sip.uk.oracle.com"
gateway_alarm_severity = 3
gateway_alarm_message = "Unable to Contact Gateway"
heartbeat_release_cause = 127

### Configuration for receiving keepalive messages ###

restart_charging = true
heartbeat_receive_timeout = 5
charging_restart_svc_key = 2
reply_to_options_heartbeat = false

### Configuration for parsing the Media attributes in the SDP ###

media_mapping = [ { sdp_params="audio:send,video:send", capability=1 },
                  { sdp_params="audio:sendrecv",        capability=2 },
                  { sdp_params="audio:",                capability=3 },
                  { sdp_params="*:*",                   capability=4 } ]
allow_unmapped_media = true
media_mapping_alarm_severity = 3
media_mapping_alarm_message = "Media Mapping Error"

media_change_no_dp = "REJECT_MEDIA"

### UPDATE message handling configuration ###

update_message_handling = "REJECT"

### Call-ID Retrieval configuration ###
msc_address="987654321"
msc_noa=4
msc_plan=1
call_id_alarm_severity = 1
call_id_alarm_message = "Call ID Greater than 64 Characters"
uniqueCallId="ESGSCA"

### Hold Message configuration ###

inactive_media_hold = true
allow_overlap_invite=true
etc_append_cdpn=false

### Number translation ###

trans_mapping= [ { domain = "abc.com" , prefix = "01" }
                 { domain = "def.com" , prefix = "02" }
                 { domain = "ghi.com" , prefix = "03" }
               ]

trans_behaviour = [ "DISPLAYNAME","URI" ]
strip_matched_prefix=false
default_prefix = ""
replace_plus=false
insert_plus=false

denorm_mapping= [ { prefix = "44" , remove_chars = "2" , add_chars = "" }
                  { prefix = "*"  , remove_chars = "2" , add_chars = "+" }
                ]

norm_mapping=   [ { prefix = "0"  , remove_chars = "1" , add_chars = "44" }
                   { prefix = "*"  , remove_chars = "2" , add_chars = "0" }
```

```
                    ]

    hashEncodeChar=' '
    starEncodeChar='f'
    always_trans_map=true

    ### Configuration for ENUM URI Support ###

    enum_data_profile_tag = 0
    enum_service_type = "E2U+sip"
    enum_data_type = "AUS"|"URI"|"NAPTR"|"DYN"|"DYNAMIC"
    enum_enabled = true

    ### Configuration for INVITE messages ###

    p_asserted_identity=false
    call_init_use_reinvite=false
    call_init_send_empty_address=false
    call_init_a_include_cap4_xml=false
    call_init_b_include_cap4_xml=false
    call_init_a_cap4_use_suppress_t_csi = false

}
```

High level parameters

The SCA interface accepts the following high level parameters.

`b2bua`

Syntax:	`b2bua = `*`true`*`|`*`false`*
Description:	Sets whether or not the SCA handles all calls in transparent back-to-back user agent mode.
Type:	Boolean
Optionality:	Optional (default used if not set).
Allowed:	• true - all calls will be handled in b2bua mode • false - all calls will be handled in proxy mode (unless otherwise specified in a rule in **rules.nhp**.
Default:	true
Notes:	
Example:	`b2bua = true`

`cdrAgeLimit`

Syntax:	`cdrAgeLimit = `*`secs`*
Description:	Defines the maximum number of seconds that a EDR file will remain open.
Type:	Integer
Optionality:	Optional (default used if not set).
Allowed:	
Default:	600
Notes:	If the specified maximum age is reached, then a new EDR file is created.
Example:	`cdrAgeLimit = 600`

`cdrFinalDir`

Syntax:	`cdrFinalDir = "`*`dir`*`"`
Description:	The location for the final EDR files generated by the SCA.

Type:	String
Optionality:	Optional (default used if not set).
Allowed:	
Default:	"/IN/service_packages/SCA/cdr/closed"
Notes:	
Example:	`cdrFinalDir = "/var/cdr/SCA/final"`

cdrSizeLimit

Syntax:	`cdrSizeLimit =` *`num`*
Description:	Defines the maximum number of records that can be created in a single EDR file.
Type:	Integer
Optionality:	Optional (default used if not set).
Allowed:	
Default:	100000
Notes:	If the specified maximum is exceeded, then a new EDR file is created to hold the additional records.
Example:	`cdrSizeLimit = 100000`

cdrTempDir

Syntax:	`cdrTempDir = "`*`dir`*`"`
Description:	The location for the temporary EDR files generated by the SCA.
Type:	String
Optionality:	Optional (default used if not set).
Allowed:	
Default:	"/IN/service_packages/SCA/cdr/open"
Notes:	
Example:	`cdrTempDir = "/var/cdr/SCA/open"`

dns_cache_time

Syntax:	`dns_cache_time =` *`secs`*
Description:	The number of seconds between updates of the **rules.nhp** DNS cache.
Type:	Integer.
Optionality:	Optional (default used if not set).
Allowed:	A valid integer.
Default:	300
Notes:	All hostnames in the **rules.nhp** file are resolved to IP addresses when the SCA starts. dns_cache_time defines the interval for periodically updating this cached information.
Example:	`dns_cache_time = 500`

enableCDRs

Syntax:	`enableCDRs =` *`true`*`|`*`false`*
Description:	Whether the SCA produces EDRs.
Type:	Boolean
Optionality:	Optional (default used if not set).

Allowed:	true, false
Default:	true
Notes:	If this parameter is not set because you do not want the SCA to produce EDRs, then you do not need to set any other parameters relevant to EDR generation.
Example:	`enableCDRs = false`

failed_address_timeout

Syntax:	`failed_address_timeout =` *`secs`*
Description:	Defines the number of seconds that failed target addresses (in the Rules file) will remain in this state. Subsequent requests that trigger the rule will be forwarded to alternate addresses until this timer expires.
Type:	Integer.
Optionality:	Optional (default used if not set).
Allowed:	A valid integer.
Default:	300
Notes:	
Example:	`failed_address_timeout = 200`

inap_inn

Syntax:	`inap_inn =` *`true`*`|`*`false`*
Description:	Whether the Internal Network Number flag is set in the IDP sent by the SCA.
Type:	Boolean
Optionality:	Optional (default used if not set).
Allowed:	true, false
Default:	true
Notes:	
Example:	`inap_inn = false`

inap_noa_called_party

Syntax:	`inap_noa_called_party =` *`noa`*
Description:	The default nature of address to use in the IDP sent by the SCA for the called party.
Type:	Integer
Optionality:	Optional (default used if not set).
Allowed:	
Default:	4
Notes:	
Example:	`inap_noa_called_party = 4`

inap_noa_calling_party

Syntax:	`inap_noa_calling_party =` *`noa`*
Description:	The default nature of address to use in the IDP sent by the SCA for the calling party.
Type:	Integer
Optionality:	Optional (default used if not set).
Allowed:	

Default:	4
Notes:	
Example:	`inap_noa_calling_party = 4`

`inap_noa_redirecting_party`

Syntax:	`inap_noa_redirecting_party =` *`noa`*
Description:	The default nature of address to use in the IDP sent by the SCA for the redirecting party.
Type:	Integer
Optionality:	Optional (default used if not set).
Allowed:	
Default:	4
Notes:	
Example:	`inap_noa_redirecting_party = 4`

`inap_numplan`

Syntax:	`inap_numplan =` *`value`*
Description:	The numbering plan value used in the IDP sent by the SCA.
Type:	Integer
Optionality:	Optional (default used if not set).
Allowed:	A numeric value.
Default:	131
Notes:	
Example:	`inap_numplan = 2`

`inap_pres`

Syntax:	`inap_pres =` *`value`*
Description:	The presentation value used in the IDP sent by the SCA.
Type:	Integer
Optionality:	Optional (default used if not set).
Allowed:	A numeric value.
Default:	0
Notes:	
Example:	`inap_pres = 1`

`inap_scr`

Syntax:	`inap_scr =` *`value`*
Description:	The screening value used in the IDP sent by the SCA.
Type:	Integer
Optionality:	Optional (default used if not set)
Allowed:	A numeric value
Default:	3
Notes:	
Example:	`inap_scr = 1`

`invite_failover_only`

Syntax:	`invite_failover_only = `*`true`*`|`*`false`*
Description:	If a destination address is unreachable and this parameter is set to true, the SCA will try an alternate address (if available) when forwarding an INVITE request..
Type:	Boolean
Optionality:	Optional (default used if not set).
Allowed:	• true - only try an alternate address when forwarding an INVITE request • false - try alternate addresses for any request.
Default:	true
Notes:	
Example:	`invite_failover_only = false`

`load_balancing_enabled`

Syntax:	`load_balancing_enabled = `*`true`*`|`*`false`*
Description:	Determines whether or not requests should be load balanced to multiple destinations if a list of URIs is defined in a rule.
Type:	Boolean
Optionality:	Optional (default used if not set).
Allowed:	true, false
Default:	true
Notes:	
Example:	`load_balancing_enabled = false`

`local_contact`

Syntax:	`local_contact = `*`true`*`|`*`false`*
Description:	Whether contact should be set as sca IP and Port.
Type:	Boolean
Optionality:	Optional (default used if not set).
Allowed:	true, false
Default:	false
Notes:	Example: If set to true: `200ok Contact: <sip:19425@10.11.188.19:7070>` becomes `Contact: <sip:10.11.188.19:7070>`
Example:	`local_contact = false`

`local_ip`

Syntax:	`local_ip = "`*`ipaddr`*`"`
Description:	Defines the IP address of the local machine that is inserted into Via: header for outgoing requests
Type:	String
Optionality:	Mandatory
Allowed:	A valid IP address
Default:	
Notes:	

Example:	`local_ip = "127.0.0.1"`

`media_change_no_dp`

Syntax:	`media_change_no_dp = "`*`action`*`"`
Description:	Specifies what action to take if a media change is received but the Service_Changed detection point is not set. This is usually set by a node in the control plan.
Type:	String
Optionality:	Optional (default used if not set).
Allowed:	• ALLOW_MEDIA - allow through the message with the new media • CLEAR_MEDIA - terminate the call • REJECT_MEDIA - respond to the message with a 415 Unsupported Media Type response
Default:	REJECT_MEDIA
Notes:	
Example:	`media_change_no_dp = "REJECT_MEDIA"`

`noin_dropcall`

Syntax:	`noin_dropcall = `*`true`*`|`*`false`*
Description:	Used when the SCA is required to forward a request and then "get out of the loop".
Type:	Boolean
Optionality:	Optional (default used if not set).
Allowed:	Either: • true - a call will be cleared internally after the first request is forwarded, or • false - the SCA will remain in the loop for all messages within a call.
Default:	false
Notes:	The SCA must be operating in proxy mode when this option is set to true. The control plan must use a termination node that exits as soon as the call is connected. The SCA will then forward the request, but it will not add itself into the Record-Route list. This means that subsequent requests within the same call will not be routed via the SCA.
Example:	`noin_dropcall = true`

`propagate_b_error`

Syntax:	`propagate_b_error = `*`true`*`|`*`false`*
Description:	Determines what error code is returned to the A-leg when the B-leg returns an error.
Type:	Boolean
Optionality:	Optional (default used if not set).
Allowed:	• true - return the B-party error code to the A-party unchanged, or • false - map the release cause returned from ACS to a SIP error code and return this error code to the A-party.
Default:	false
Notes:	By default, the SCA notifies the IN of an error and the call model can then change the error code by setting a release cause. If this parameter is set, then the B-leg error code will always be returned to the A-leg unchanged.

Example:	`propagate_b_error = true`

oracleUserAndPassword

Syntax:	`oracleUserAndPassword = "`*`user/password`*`"`
Description:	The login ID for the database on the SMS node. The remoteCommanderUser utility uses this ID to log in to the database when setting the password for the SCA Remote Commander.
Type:	String
Optionality:	Optional (default used if not set)
Allowed:	
Default:	/
Notes:	
Example:	`oracleUserAndPassword = "/"`

registrar

Contains the registrar cache size and contact expiry parameters within {}. See *registrar section* (on page 54).

rem_comm_port

Syntax:	`rem_comm_port = `*`port`*
Description:	The port on which the remote commander is listening.
Type:	Integer
Optionality:	Optional (default used if not set)
Allowed:	A valid port number.
Default:	3615
Notes:	
Example:	`rem_comm_port = 3615`

replace_diversion_header

Syntax:	`replace_diversion_header = `*`true|false`*
Description:	Determines how a redirecting number returned from ACS is handled. It will perform one of the following: • Be added to a list of Diversion headers • Replace any existing Diversion headers
Type:	Boolean
Optionality:	Optional (default used if not set).
Allowed:	• true - replace any existing Diversion headers with a single Diversion header containing the redirecting number • false - add a new Diversion header containing the redirecting number to an existing list of Diversion headers.
Default:	false
Notes:	
Example:	`replace_diversion_header = true`

rules

Syntax:	`rules = "`*`dir/file`*`"`
Description:	The location of the Next Hop Processing rules file.

Type:	String
Optionality:	Optional (default used if not set).
Allowed:	
Default:	"/IN/service_packages/SCA/etc/rules.nhp"
Notes:	
Example:	`rules = "/IN/service_packages/sca_rules.nhp"`

`uniqueCallId`

Syntax:	`uniqueCallId = "`*`value`*`"`
Description:	The text part of the SCA call ID. SCA generates the call ID when initiating a new call leg. For example, if you set `uniqueCallID` to "ESGSCA" and the first leg of a call has ID 143302914@10.170.23.22, SCA generates call ID ESGSCA143302914@10.170.23.22 for the next call leg.
Type:	String
Optionality:	Optional (default used if not set)
Allowed:	
Default:	ESGSCA
Notes:	
Example:	`uniqueCallId = "ESGSCA"`

`update_diversion_header`

Syntax:	`update_diversion_header = true|false`
Description:	This parameter controls the Diversion header change. Useful when you want to retain the existing Diversion header.
Type:	Boolean
Optionality:	Optional (default used if not set)
Allowed:	• true - retains default behaviour. • false - restricts the replacement of any existing Diversion headers with new one.
Default:	true
Notes:	
Example:	`update_diversion_header = true`

`use_ALegCallID`

Syntax:	`use_ALegCallID = `*`true|false`*
Description:	Whether to use the custom header for B-legs to assist in debugging with snoop output.
Type:	Boolean
Optionality:	Optional (default used if not set).
Allowed:	true, false
Default:	true
Notes:	
Example:	`use_ALegCallID = true`

`usePAssertedID`

Syntax:	`usePAssertedID = `*`true`*`|`*`false`*
Description:	Sets whether or not the SCA should use the number in the P-Asserted_ID header (if present) as the calling party number in the IDP sent to the IN.
Type:	Boolean
Optionality:	Optional (default used if not set).
Allowed:	Either: • true - extract the calling party number from the P-Asserted-ID header, or • false - extract the calling party number from the From header.
Default:	false
Notes:	
Example:	`usePAssertedID = true`

registrar section

Here is an example of the `registrar` section configuration.

```
registrar = {
    cacheSize = 0
    defaultExpiry = 3600
}
```

The registrar section of the SCA configuration supports the following parameters.

`cacheSize`

Syntax:	`cacheSize= `*`size`*
Description:	The cache size for the registrar.
Type:	Integer
Optionality:	Optional (default used if not set).
Allowed:	
Default:	0 (Unlimited)
Notes:	
Example:	`cacheSize= 0`

`defaultExpiry`

Syntax:	`defaultExpiry = seconds`
Description:	The contact expiry time (in seconds) in the registrar.
Type:	Integer
Optionality:	Optional (default used if not set).
Allowed:	
Default:	3600
Notes:	
Example:	`paraMeter = 3600`

Remote Party ID Header configuration

The configuration section for Remote Party ID Header supports the following parameters.

`cf_use_cdpn_from_request`

Syntax:	`cf_use_cdpn_from_request =` *`true`*`|`*`false`*
Description:	Set this parameter to true to set the cdpn from the Request field
Type:	Boolean
Optionality:	Optional (default used if not set).
Allowed:	true, false
Default:	false
Notes:	If the REQUEST: username and the TO: username are different and the DIVERSION header is present then SIP call forwarding is in use. Therefore the ACS IDP cdpn uses the REQUEST: username, the cgpn uses the FROM: username. cf means Calling forwarding
Example:	`cf_use_cdpn_from_request = false`

`include_pai_tel_header`

Syntax:	`include_pai_tel_header =` *`true`*`|`*`false`*
Description:	If enabled the tel URI p-asserted id header will be generated/translated.
Type:	Boolean
Optionality:	Optional (default used if not set).
Allowed:	
Default:	false
Notes:	Used only when `p_asserted_identity_trans` is ALWAYS or TRANSPARENT.
Example:	`include_pai_tel_header = false`

`include_rpi_id_type`

Syntax:	`include_rpi_id_type =` *`true`*`|`*`false`*
Description:	If enabled the id-type token will be added to generated remote-party-id headers.
Type:	Boolean
Optionality:	Optional (default used if not set).
Allowed:	true, false
Default:	false
Notes:	
Example:	`include_rpi_id_type = false`

`include_rpi_privacy`

Syntax:	`include_rpi_privacy =` *`true`*`|`*`false`*
Description:	If enabled the privacy token will be added to generated remote-party-id headers based on the calling party presentation parameter in the call model.
Type:	Boolean
Optionality:	Optional (default used if not set).
Allowed:	true, false
Default:	false
Notes:	
Example:	`include_rpi_privacy = false`

`include_rpi_pty_type`

Syntax:	`include_rpi_pty_type =` *`true`*`|`*`false`*
Description:	If enabled the party token will be added to generated remote-party-id headers.
Type:	Boolean
Optionality:	Optional (default used if not set).
Allowed:	true, false
Default:	false
Notes:	
Example:	`include_rpi_pty_type = false`

`include_rpi_screen`

Syntax:	`include_rpi_screen =` *`true`*`|`*`false`*
Description:	If enabled the screen token will be added to generated remote-party-id headers based on the calling party screening parameter in the call model
Type:	Boolean
Optionality:	Optional (default used if not set).
Allowed:	true, false
Default:	false
Notes:	
Example:	`include_rpi_screen = false`

`p_asserted_identity_trans`

Syntax: `p_asserted_identity_trans = "`*`value`*`"`

Description:

Type: String

Optionality: Optional (default used if not set).

Allowed:

- "ALWAYS"
- "TRANSPARENT"
- "NEVER"

Default: "NEVER"

Notes: If set to "ALWAYS":

- When the incoming INVITE contains a p-asserted-identity header the header uri is to be set to match the from header uri in the outgoing INVITE message.
- When the incoming INVITE does not contain a p-asserted-identity header a new header will be configured according to the parameters below and added to the outgoing INVITE, the header will contain a uri matching the From header uri as a minimum.

If set to "TRANSPARENT"

- When the incoming INVITE contains a p-asserted-identity header the header uri is to be set to match the from header uri in the outgoing INVITE message.
- When the incoming INVITE does not contain a p-asserted-identity header no action will be taken.

If set to "NEVER"

- No attempt to translate or generate p-asserted-identity headers

Example:	`p_asserted_identity_trans = "NEVER"`

`remotePartyIdTrans`

Syntax: `remotePartyIdTrans = "`*`value`*`"`

Description:

Type: String

Optionality: Optional (default used if not set).

Allowed:

- "ALWAYS"
- "TRANSPARENT"
- "NEVER"

Default: "NEVER"

Notes: If set to "ALWAYS":

- When the incoming INVITE contains a remote-party-id header with the party token absent or set to calling the header uri is be set to match the from header uri in the outgoing INVITE message.
- When the incoming INVITE does not contain a remote-party-id header a new header will be configured according to the parameters below and added to the outgoing INVITE, the header will contain a uri matching the From header uri as a minimum

If set to "TRANSPARENT"

- When the incoming INVITE contains a remote-party-id header with the party token absent or set to calling the header uri is be set to match the from header uri in the outgoing INVITE message.
- When the incoming INVITE does not contain a remote-party-id header no action will be taken

If set to "NEVER"

- No attempt to translate or generate remote-party-id headers

Example: `remotePartyIdTrans = "NEVER"`

`rpi_addressNotAvailable`

Syntax: `rpi_addressNotAvailable = "`*`value`*`"`

Description: Maps Set Indicator feature node presentation restricted value to remote party id header privacy tag value.

Type: String

Optionality: Optional (default used if not set).

Allowed:

Default: "uri"

Notes:

Example: `rpi_addressNotAvailable = "uri"`

`rpi_id_type`

Syntax: `rpi_id_type = "`*`value`*`"`

Description: Defines the value that will be given to id_type token.

Type: String

Optionality: Optional (default used if not set).

Allowed:	• "subscriber" • "user" • "alias" • "return" • "term"
Default:	"user"
Notes:	Used only if `include_rpi_id_type` parameter is true.
Example:	`rpi_id_type = "user"`

rpi_network_provided

Syntax:	`rpi_network_provided = "`*`value`*`"`
Description:	Maps Set Indicator feature node screening value to remote party id header privacy tag value.
Type:	String
Optionality:	Optional (default used if not set).
Allowed:	
Default:	"no"
Notes:	
Example:	`rpi_network_provided = "no"`

rpi_presentation_allowed

Syntax:	`rpi_presentation_allowed = "`*`value`*`"`
Description:	Maps Set Indicator feature node presentation restricted value to remote party id header privacy tag value.
Type:	String
Optionality:	Optional (default used if not set).
Allowed:	
Default:	"full"
Notes:	
Example:	`rpi_presentation_allowed = "full"`

rpi_presentationRestricted

Syntax:	`rpi_presentationRestricted = "`*`value`*`"`
Description:	Maps Set Indicator feature node presentation restricted value to remote party id header privacy tag value.
Type:	String
Optionality:	Optional (default used if not set).
Allowed:	
Default:	"off"
Notes:	
Example:	`rpi_presentationRestricted = "off"`

rpi_pty_type

Syntax:	`rpi_pty_type = "`*`value`*`"`
Description:	Defines the value that will be given to party token.
Type:	String

Optionality:	Optional (default used if not set).
Allowed:	• "calling" • "called"
Default:	"calling"
Notes:	Used only if `include_rpi_pty_type` parameter is true.
Example:	`rpi_pty_type = "calling"`

rpi_spare

Syntax:	`rpi_spare = "`*`value`*`"`
Description:	Maps Set Indicator feature node presentation restricted value to remote party id header privacy tag value.
Type:	String
Optionality:	Optional (default used if not set).
Allowed:	
Default:	"name"
Notes:	
Example:	`rpi_spare = "name"`

rpi_user_not_verified

Syntax:	`rpi_user_not_verified = "`*`value`*`"`
Description:	Maps Set Indicator feature node screening value to remote party id header privacy tag value.
Type:	String
Optionality:	Optional (default used if not set).
Allowed:	
Default:	"no"
Notes:	
Example:	`rpi_user_not_verified = "no"`

rpi_user_verified_failed

Syntax:	`rpi_user_verified_failed = "`*`value`*`"`
Description:	Maps Set Indicator feature node screening value to remote party id header privacy tag value.
Type:	String
Optionality:	Optional (default used if not set).
Allowed:	
Default:	"no"
Notes:	
Example:	`rpi_user_verified_failed = "no"`

rpi_user_verified_passed

Syntax:	`rpi_user_verified_passed = "`*`value`*`"`
Description:	Maps Set Indicator feature node screening value to remote party id header privacy tag value.
Type:	String

Optionality:	Optional (default used if not set).
Allowed:	
Default:	"yes"
Notes:	
Example:	`rpi_user_verified_passed = "yes"`

update_rpi_privacy

Syntax:	`update_rpi_privacy =` *`true`*`|`*`false`*
Description:	If enabled the privacy token will be modified in existing remote-party-id headers based on the calling party presentation parameter in the call model.
Type:	Boolean
Optionality:	Optional (default used if not set).
Allowed:	true, false
Default:	false
Notes:	
Example:	`update_rpi_privacy = false`

update_rpi_screen

Syntax:	`update_rpi_screen =` *`true`*`|`*`false`*
Description:	If enabled the screen token will be modified in existing remote-party-id headers based on the calling party screening parameter in the call model.
Type:	Boolean
Optionality:	Optional (default used if not set).
Allowed:	true, false
Default:	false
Notes:	
Example:	`update_rpi_screen = false`

heartbeating to a gateway configuration

The configuration section for heartbeating in the SCA configuration supports the following parameters.

gateway_alarm_message

Syntax:	`gateway_alarm_message = "`*`string`*`"`
Description:	Sets the message in the alarm raised as a result of no response to an OPTIONS message to the gateway.
Type:	String
Optionality:	Optional
Allowed:	
Default:	"Detected Gateway Failure"
Notes:	
Example:	`gateway_alarm_message = "Unable to Contact Gateway"`

gateway_alarm_severity

Syntax:	`gateway_alarm_severity =` *`sev`*
Description:	Sets the severity of the alarm raised as a result of no response to an OPTIONS message to the gateway.

Type:	Integer
Optionality:	Optional
Allowed:	• 0 (NOTICE) • 1 (WARNING) • 2 (ERROR) • 3 (CRITICAL) • 4 (CLEAR)
Default:	3
Notes:	If this value is not defined, the header will not be present.
Example:	`gateway_alarm_severity = 3`

heartbeat_destination

Syntax:	`heartbeat_destination= "`*`rule`*`"`
Description:	The destination of the heartbeat OPTIONS message.
Type:	String
Optionality:	Mandatory if `heartbeat_send_interval` is greater than 0
Allowed:	
Default:	
Notes:	The value must be the name of a rule defined in **rules.nhp** (not a URI).
Example:	`heartbeat_destination = "gateway_rule"`

heartbeat_from_address

Syntax:	`heartbeat_from_address = "`*`address`*`"`
Description:	Sets the URI part of the From header in the OPTIONS heartbeat message.
Type:	String
Optionality:	Mandatory if `heartbeat_send_interval` is greater than 0
Allowed:	
Default:	
Notes:	
Example:	`heartbeat_from_address = "uas.uk.oracle.com"`

heartbeat_p_asserted_id

Syntax:	`heartbeat_p_asserted_id = `*`id`*
Description:	Sets the entire P-Asserted-Identity header in the OPTIONS heartbeat message, including the username.
Type:	String
Optionality:	Mandatory if `heartbeat_send_interval` is greater than 0
Allowed:	
Default:	
Notes:	Skip the quotation marks to include the value in the outgoing message for the username
Example:	`heartbeat_p_asserted_id = "\"uk.oracle.com\" <sip:12345678900@sip.uk.oracle.com>"`

heartbeat_p_charging_vector

Syntax:	`heartbeat_p_charging_vector = "`*`pcv`*`"`
Description:	Sets the entire P-Charging-Vector header in the OPTIONS heartbeat message.
Type:	String
Optionality:	Optional
Allowed:	
Default:	
Notes:	If this value is not defined, the header will not be present.
Example:	`heartbeat_p_charging_vector = "icid-value=prepaid123456789012345;orig-ioi=prepaid.sip.uk.oracle.com"`

heartbeat_release_cause

Syntax:	`heartbeat_release_cause = `*`code`*
Description:	Sets the release cause that is written to the BE EDR in the RELC field, when no response is received to an OPTIONS message.
Type:	Integer
Optionality:	Optional
Allowed:	0 to 127 (inclusive)
Default:	55
Notes:	
Example:	`heartbeat_release_cause = 127`

heartbeat_request_uri

Syntax:	`heartbeat_request_uri = "`*`uri`*`"`
Description:	Sets the URI part of the OPTIONS heartbeat message request line.
Type:	String
Optionality:	Mandatory if `heartbeat_send_interval` is greater than 0
Allowed:	
Default:	
Notes:	
Example:	`heartbeat_request_uri="gateway.uk.oracle.com"`

heartbeat_route

Syntax:	`heartbeat_route = "route"`
Description:	Sets the URI part of the Route header in the OPTIONS heartbeat message.
Type:	String
Optionality:	Mandatory if `heartbeat_send_interval` is greater than 0
Allowed:	
Default:	
Notes:	
Example:	`heartbeat_route="gateway.uk.oracle.com"`

`heartbeat_send_interval`

Syntax:	`heartbeat_send_interval = seconds`
Description:	The frequency in seconds to send a heartbeat message to the gateway.
Type:	Integer
Optionality:	Optional (default used if not set).
Allowed:	
Default:	0
Notes:	The timer is restarted when a response is received or the original message times-out. If set to 0, heartbeats to the gateway and all further heartbeating options are disabled. If a heartbeat message fails, all SIP interactions will be moved to the Charging Void state and charging will be stopped.
Example:	`heartbeat_send_interval = 30`

`heartbeat_send_timeout`

Syntax:	`heartbeat_send_timeout = `*`seconds`*
Description:	The default timeout period for receiving a response to the 64 OPTIONS messages.
Type:	Integer
Optionality:	Optional (default used if not set).
Allowed:	
Default:	0 (uses 64*T1)
Notes:	This value will override the default timeout of 64*T1 in the ESC section that is normally used by the heartbeat OPTIONS message.
Example:	`heartbeat_send_timeout = 10`

`heartbeat_to_address`

Syntax:	`heartbeat_to_address = "`*`address`*`"`
Description:	Sets the URI part of the To header in the OPTIONS heartbeat message.
Type:	String
Optionality:	Mandatory if `heartbeat_send_interval` is greater than 0
Allowed:	
Default:	
Notes:	
Example:	`heartbeat_to_address="gateway.uk.oracle.com"`

keepalive messages configuration

The configuration section for keepalive messages in the SCA configuration supports the following parameters.

`charging_restart_svc_key`

Syntax:	`charging_restart_svc_key = `*`key`*
Description:	Defines the service key on which the IDP used to restart charging is sent.
Type:	Integer
Optionality:	Optional (default used if not set).
Allowed:	

Default:	0
Notes:	If this value is not defined or is set to 0, the service key used to send the original IDP will be used.
Example:	`charging_restart_svc_key = 2`

`heartbeat_receive_timeout`

Syntax:	`heartbeat_receive_timeout =` *`seconds`*
Description:	The timeout period for receiving a keepalive message.
Type:	Integer
Optionality:	Optional (default used if not set).
Allowed:	
Default:	0
Notes:	Setting the value to 0 will disable this feature.
Example:	`heartbeat_receive_timeout = 0`

`reply_to_options_heartbeat`

Syntax:	`reply_to_options_heartbeat=` *`true`*`|`*`false`*
Description:	Send a 200-OK response to any OPTIONS messages received mid-call.
Type:	Boolean
Optionality:	Optional (default used if not set).
Allowed:	true, false
Default:	false
Notes:	Use this for interception OPTIONS heartbeat messages and responding to them. Can be used with `heartbeat_receive_timeout`.
Example:	`reply_to_options_heartbeat= false`

`restart_charging`

Syntax:	`restart_charging=` *`true`*`|`*`false`*
Description:	Determines whether charging will be restarted when a 200--OK response is received from the B-Party to a keepalive message from the A-party.
Type:	Boolean
Optionality:	Optional (default used if not set).
Allowed:	true, false
Default:	false
Notes:	
Example:	`restart_charging = true`

media attributes in SDP section

The media attributes section of the SCA configuration supports the following parameters.

`allow_unmapped_media`

Syntax:	`allow_unmapped_media =` *`true`*`|`*`false`*
Description:	Whether an INVITE with SDP information with no associated mapping should be translated to an IDP with no bearer capability information.
Type:	Boolean

Optionality: Optional (default used if not set).

Allowed:

true	Translate SDP media details to IDP with no bearer capability
false	Raise alarm and reject the call

Default: false

Notes:

Example: `allow_unmapped_media = true`

media_mapping

Syntax:

```
media_mapping = [
     { sdp_params="str", capability=int }
     [...]
]
```

Description: The media_mapping table maps media fields (m= and a=) in the SDP of an incoming INVITE to Bearer Capability information which will be sent in the IDP.

Type: Array

Optionality: Optional

Allowed:

Default:

Notes: For more details, see *Media mapping table* (on page 31).
A single row in the table must contain one sdp_params item and one capability item, for example:

```
{ sdp_params="audio:send,video:recv", capability=1 }
```

will map an sdp with "m=audio,a=send,m=audio,a=recv" to a bearer capability of 1.
Media type attributes can be combined, for example:

```
{ sdp_params="audio:send,audio:recv", capability=1 }
```

will match an sdp with "m=audio,a=send,a=recv"

The following bearer capability parameters can also be defined in a rule:

- protocol (default 0xff)
- transfer_rate
- transfer_mode
- coding_standard

Using '*' as a media type or attribute in the sdp_params will match anything.

Using a blank attribute will match only media types with no attributes ("audio:").

Example:

```
media_mapping = [ { sdp_params="audio:send,video:send", capability=1 },
                  { sdp_params="audio:sendrecv",        capability=2 },
                  { sdp_params="audio:",                capability=3 },
                  { sdp_params="*:*",                   capability=4 } ]
```

media_mapping_alarm_message

Syntax: `media_mapping_alarm_message = "str"`

Description: The message in the alarm raised as a result of no mapping being found for information in the SDP of an incoming INVITE.

Type: String

Optionality: Optional

Allowed:	
Default:	"No media/attribute mappings found"
Notes:	
Example:	`media_mapping_alarm_message = "Media Mapping Error"`

`media_mapping_alarm_severity`

Syntax:	`media_mapping_alarm_severity =` *`sev`*
Description:	The severity of the alarm raised as a result of no mapping being found for information in the SDP of an incoming INVITE.
Type:	Integer
Optionality:	Optional (default used if not set).
Allowed:	0 Notice 1 Warning 2 Error 3 Critical 4 Clear
Default:	2
Notes:	
Example:	`media_mapping_alarm_severity = 3`

UPDATE message handling section

The UPDATE message section of the SCA configuration supports the following parameter.

`update_message_handling`

Syntax:	`update_message_handling = "`*`action`*`"`
Description:	Sets the behavior of the SCA when an UPDATE message is received for a call that is not in the Charging Void state.
Type:	String
Optionality:	Optional (default used if not set).
Allowed:	PROXY Send the UPDATE message to its destination. IGNORE Swallow the message. REJECT Return a SIP_METHOD_NOT_ALLOWED message.
Default:	PROXY
Notes:	
Example:	`update_message_handling = "REJECT"`

Call ID Retrieval section

The Call ID section of the SCA configuration supports the following parameters.

`call_id_alarm_message`

Syntax:	`call_id_alarm_message =` *`string`*
Description:	Sets the message in the alarm raised if the Call ID is greater than 64 characters..
Type:	String
Optionality:	Optional
Allowed:	

Default:	"Call ID Truncated"
Notes:	
Example:	`call_id_alarm_message = "Call ID Greater than 64 Characters"`

`call_id_alarm_severity`

Syntax:	`call_id__alarm_severity =` *`sev`*
Description:	Sets the severity of the alarm raised if the Call ID is greater than 64 characters (the maximum that can be sent in the call reference field in the IDP).
Type:	Integer
Optionality:	Optional
Allowed:	• 0 (NOTICE) • 1 (WARNING) • 2 (ERROR) • 3 (CRITICAL) • 4 (CLEAR)
Default:	1 (WARNING)
Notes:	
Example:	`call_id__alarm_severity = 1`

`msc_address`

Syntax:	`msc_address = "`*`addr`*`"`
Description:	Sets the MSC Address in the IDP.
Type:	Integer
Optionality:	Optional
Allowed:	
Default:	not set
Notes:	A matching MSC address can be entered in the **eserv.config** file to write the Call ID in ASCII form.
Example:	`msc_address = "987654321"`

`msc_noa`

Syntax:	`msc_noa =` *`noa`*
Description:	Sets the nature of address for the parameter `msc_address`
Type:	Integer
Optionality:	Optional
Allowed:	
Default:	0
Notes:	
Example:	`msc_noa = 4`

`msc_plan`

Syntax:	`msc_plan =` *`int`*
Description:	Sets the plan of address for the parameter `msc_address`
Type:	Integer
Optionality:	Optional

Allowed:	
Default:	0
Notes:	
Example:	`msc_plan = 1`

Hold message section

The Hold message section of the SCA configuration supports the following parameters.

`allow_overlap_invite`

Syntax:	`allow_overlap_invite =` *`true`*`|`*`false`*
Description:	An overlap INVITE may be sent when a party is dialing slowly. A re-INVITE may be sent before any response to the original INVITE is received.
Type:	Boolean
Optionality:	Optional (default used if not set).
Allowed:	• false - respond to any overlap INVITEs with a 491 Request Pending response • true - forward overlap INVITEs to the B-leg and respond to the original INVITE with a 484 Address Incomplete message.
Default:	false
Notes:	
Example:	`allow_overlap_invite = true`

`etc_append_cdpn`

Syntax:	`etc_append_cdpn =` *`true`*`|`*`false`*
Description:	If an establish temporary connection message is returned to the SCA from ACS then the called party number will be forwarded in the To header of the outgoing request. If etc_append_cdpn is set to true, then the original called party number will be appended to the To header.
Type:	Boolean
Optionality:	Optional (default used if not set).
Allowed:	• true - append the original called party number to the To header • false - do not append the original called party number
Default:	false
Notes:	
Example:	`etc_append_cdpn = true`

`inactive_media_hold`

Syntax:	`inactive_media_hold =` *`true`*`|`*`false`*
Description:	Determines the format of the SDP information in a re-INVITE used by the SCA to put a call on hold.
Type:	Boolean
Optionality:	Optional (default used if not set).
Allowed:	true, false
Default:	false
Notes:	By default, the hold messages sent by the SCA have the `c=` field in the SDP set to "IN IP4 0.0.0.0".

	If this option is set to true, the IDP will instead contain a single `m=` parameter set to "inactive".
Example:	`inactive_media_hold = true`

Number translation

The number translation section of the SCA configuration supports the following parameters.

always_trans_map

Syntax:	`always_trans_map = `*`true`*`|`*`false`*
Description:	Always run trans_mapping.
Type:	Boolean
Optionality:	Optional (default used if not set).
Allowed:	true, false
Default:	true
Notes:	If set to false, then the translation mapping will only be done if the IN has changed the number.
Example:	`always_trans_map = true`

default_prefix

Syntax:	`default_prefix = "`*`pref`*`"`
Description:	Defines the default prefix to add to numbers returned by the IN application, if the prefix is not defined in the trans_mapping table.
Type:	String
Optionality:	
Allowed:	
Default:	""
Notes:	
Example:	`default_prefix = ""`

denorm_mapping

List of prefixes and characters to be removed or added to numbers received from ACS. For details, see the *denorm_mapping section* (on page 72).

hashEncodeChar

Syntax:	`hashEncodeChar = '`*`char`*`'`
Description:	Defines the hexadecimal character to use to replace any hash (#) characters when translating a SIP URI to an E.164 number (before sending an IDP to ACS.
Type:	Hexadecimal string
Optionality:	Optional.
Allowed:	• '0' to 'F' • ' ' - this will remove any # characters. There must be a space between the single quote marks
Default:	None
Notes:	
Example:	`hashEncodeChar = '1'`

`insert_plus`

Syntax:	`insert_plus =` *`true`*`|`*`false`*
Description:	Determines whether or not a "+" character should be prepended to a URI if the E.164 number returned from ACS has a NoA of 4 (international).
Type:	Boolean
Optionality:	Optional (default used if not set).
Allowed:	true, false.
Default:	false
Notes:	
Example:	`insert_plus = true`

`norm_mapping`

List of prefixes and characters to be removed or added to numbers sent to ACS. For details, see *norm_mapping section* (on page 73).

`replace_plus`

Syntax:	`replace_plus =` *`true`*`|`*`false`*
Description:	Determines whether or not a "+" character preceding a URI received in a request should be replaced with "00" before sending an IDP.
Type:	Boolean
Optionality:	Optional (default used if not set).
Allowed:	true, false
Default:	false
Notes:	
Example:	`replace_plus = true`

`starEncodeChar`

Syntax:	`starEncodeChar = '`*`char`*`'`
Description:	Defines the hexadecimal character to use to replace any star (*) characters when translating a SIP URI to an E.164 number (before sending an IDP to ACS.
Type:	Hexadecimal string
Optionality:	Optional.
Allowed:	• '0' to 'F' • ' ' - this will remove any * characters. There must be a space between the single quote marks.
Default:	None
Notes:	
Example:	`starEncodeChar = 'A'`

`strip_matched_prefix`

Syntax:	`strip_matched_prefix =` *`true`*`|`*`false`*
Description:	Determines whether or not the prefix that is matched using trans_mapping should be removed.
Type:	Boolean
Optionality:	Optional (default used if not set).
Allowed:	true, false

Default:	false
Notes:	
Example:	`strip_matched_prefix = true`

`trans_mapping`

List of domains and prefixes within [] for mapping URIs to E.164 numbers and E.164 numbers to URIs. For details, see *trans_mapping section* (on page 71).

`trans_behaviour`

Syntax:	`trans_behaviour = ["`*`displayname`*`","`*`URI`*`"]`
Description:	Defines the behavior of the trans_mapping table. The specified value(s) will be replaced with the E.164 number. By default the display name and username part of the URI is replaced.
Type:	Array
Optionality:	Optional (default used if not set).
Allowed:	• "DISPLAYNAME","URI" • "URI"
Default:	"DISPLAYNAME","URI"
Notes:	For an example, refer to the trans_mapping section.
Example:	`trans_behaviour =` ["DISPLAYNAME","URI"]

trans_mapping section

The trans_mapping section maps E.164 numbers returned from ACS to domain names. If a mapping is not found then the configured default_domain is used.

Trans_mapping performs the following actions:

- By default the matched prefix is stripped from the number. This action can be disabled by setting the strip_match_prefix parameter to false
- If the returned E.164 number has a NoA value of 4 (international), then a "+" can optionally be added. A "+" is added when the insert_plus parameter is set to true
- By default both the display name and username part of the URI are replaced with the translated E.164 number. This can be changed using the `trans_behaviour` parameter
- Numbers are normalized or denormalized according to the configuration defined in the norm_mapping and denorm_mapping sections.

Example

Here is an example `trans_mapping` section configuration.

```
trans_mapping = [
    {domain = "abc.com", prefix = "01"}
    {domain = "def.com", prefix = "02"}
    {domain = "ghi.com", prefix = "03"}
]
```

The following process shows how this trans_mapping configuration is used.

1 An INVITE message is received with the following To header:
`To: User1 <sip:+44147328900@example.com>`

2 This triggers a rule that sends an IDP to ACS. According to the rule, the:
- IDP will contain the Called Party Number 441473289900
- + will be removed

- NoA of the number will be set to 4 (international)

3 The control plan:
- Changes the number to 01473123456
- Sets the NoA to national

4 The domain defined in the trans_mapping configuration for the "01" prefix is used to create a forwarded INVITE message. In this case the domain used is "abc.com" and therefore the To header is set to:

```
To: 473123456 <sip:473123456@abc.com>
```

Note:

- The "01" prefix has be removed because the `strip_matched_prefix` parameter is set to true by default
- The display name has been replaced because the `trans_behaviour` parameter is configured to do this by default, and
- `example.com` has been replaced by `abc.com` from the `trans_mapping` configuration.

Configuration

This text shows the structure of the `trans_mapping` section configuration.

```
trans_mapping = [
    {domain = "str", prefix = "str"}
    [...]
]
```

domain

Syntax:	`domain = "URI"`
Description:	Defines a URI domain.
Type:	String
Optionality:	
Allowed:	
Default:	none
Notes:	
Example:	`domain = "abc.com"`

prefix

Syntax:	`prefix= "pref"`
Description:	Defines a number prefix that maps to a URI.
Type:	String
Optionality:	
Allowed:	
Default:	none
Notes:	
Example:	`domain = "01"`

denorm_mapping section

The denorm_mapping section strips and adds characters to matched numbers received from ACS before forwarding the numbers in the outgoing SIP message.

`add_chars`

Syntax:	`add_chars = "`*`num`*`"`
Description:	The number of characters to add to the start of a number.
Type:	String
Optionality:	Optional.
Allowed:	Any number.
Default:	None
Notes:	
Example:	`add_chars = "44"`

`prefix`

Syntax:	`prefix = "`*`pref`*`"`
Description:	The prefix number to match.
Type:	String
Optionality:	Optional.
Allowed:	Any number or "*".
Default:	None
Notes:	The special character "*" can be used as a default to trigger a denormalization rule if a prefix match is not found.
Example:	`prefix = "0"`

`remove_chars`

Syntax:	`remove_chars = "`*`num`*`"`
Description:	The number of characters to remove from the start of a number.
Type:	String
Optionality:	Optional.
Allowed:	Any number.
Default:	None
Notes:	
Example:	`remove_chars = "1"`

norm_mapping section

The norm_mapping section strips and adds characters to matched numbers before sending the number to ACS in an IDP.

Here is an example of the norm_mapping section configuration.

```
norm_mapping=   [
      { prefix = "0"  , remove_chars = "1" , add_chars = "44" }
      { prefix = "*"  , remove_chars = "2" , add_chars = "0" }
]
```

`add_chars`

Syntax:	`add_chars = "`*`num`*`"`
Description:	The number of characters to add to the start of a number.
Type:	String
Optionality:	Optional.

Allowed:	Any number.
Default:	None
Notes:	
Example:	`add_chars = "44"`

`prefix`

Syntax:	`prefix = "`*`pref`*`"`
Description:	The prefix number to match.
Type:	String
Optionality:	Optional.
Allowed:	Any number, or "*".
Default:	None
Notes:	The special character "*" can be used as a default to trigger a normalization rule if a prefix match is not found.
Example:	`prefix = "0"`

`remove_chars`

Syntax:	`remove_chars = "`*`num`*`"`
Description:	The number of characters to remove from the start of a number.
Type:	String
Optionality:	Optional.
Allowed:	Any number.
Default:	None
Notes:	
Example:	`remove_chars = "1"`

ENUM URI support

The ENUM URI support section of the SCA configuration supports the following parameters.

`enum_data_profile_tag`

Syntax:	`enum_data_profile_tag = `*`tag`*
Description:	Data profile tag containing outgoing ENUM data. Used to search for ENUM data in the outgoing profile extensions.
Type:	Integer
Optionality:	Optional (default used if not set).
Allowed:	
Default:	210001 ("Enum NAPTR Temporary Store" - data tag: 210001)
Notes:	
Example:	`enum_data_profile_tag = 0`

`enum_data_type`

Syntax:	`enum_data_type = "`*`value`*`"`
Description:	Type used to interpret outgoing ENUM data.
Type:	String
Optionality:	Mandatory (If `enum_enabled` = true)

Allowed:	• "AUS" - Application Unique String (e.164 number of the form +44987654321) • "URI" - Standard URI text • "NAPTR" - One or more ENUM NAPTR records (to be converted to URIs) • "DYN"/"DYNAMIC" - Dynamic data type - type specified by the data
Default:	None
Notes:	At least one of the supported values must be used, but any mix, including all the values is allowed.
Example:	`enum_data_type = "AUS"|"URI"|"NAPTR"|"DYN"|"DYNAMIC"`

`enum_enabled`

Syntax:	`enum_enabled = `*`true|false`*
Description:	Enables (or disables) ENUM URI data extraction as part of IN processing.
Type:	Boolean
Optionality:	Optional (default used if not set).
Allowed:	true, false
Default:	false
Notes:	If true, `enum_data_type` must be configured.
Example:	`enum_enabled = true`

`enum_service_type`

Syntax:	`enum_service_type = "`*`value`*`"`
Description:	Service type filter to use when reading ENUM NAPTR data.
Type:	String
Optionality:	Optional (default used if not set).
Allowed:	
Default:	[blank]
Notes:	Suggested value: "E2U+sip"
Example:	`enum_service_type = "E2U+sip"`

INVITE messages section

The INVITE messages processed by the SCA are configured by the following parameters in the **sca.config** file:

`call_init_a_include_cap4_xml`

Syntax:	`call_init_a_include_cap4_xml = `*`true|false`*
Description:	If the sip INVITE message is for the A-Party, and `call_init_a_include_cap4_xml` is set to `true`, then send the OCSC-specific CAP4 XML to the A-Party; otherwise don't send the CAP4 XML to the A-Party.
Type:	Boolean
Optionality:	Optional (default used if not set).
Allowed:	true or false
Default:	false
Notes:	The suppress-T-CSI XML tag is included in the A-Party CAP4 XML

Example:	`call_init_a_include_cap4_xml = true`

call_init_b_include_cap4_xml

Syntax:	`call_init_b_include_cap4_xml =` *`true`*`|`*`false`*
Description:	If the sip INVITE message is for the B-Party, and a B-Party to A-Party assignment has occurred, and `call_init_b_include_cap4_xml` is set to `true`, then send the OCSC-specific CAP4 XML to the B-Party; otherwise don't send the CAP4 XML to the B-Party.
Type:	Boolean
Optionality:	Optional (default used if not set).
Allowed:	true or false
Default:	false
Notes:	The suppress-T-CSI XML tag is not included in the B-Party CAP4 XML
Example:	`call_init_include_cap4_xml = true`

call_init_a_cap4_use_suppress_t_csi

Syntax:	`call_init_a_cap4_use_suppress_t_csi =` *`true`* `|` *`false`*
Description:	During call-initiation, adds suppress-T-CSI to the CAP4/XML of the SIP body in initial A-leg INVITEs. This requires the use of MIME if both this and SDP are sent.
Type:	Boolean
Optionality:	Optional (default used if not set)
Allowed:	true, false
Default:	true
Notes:	
Example:	`call_init_a_cap4_use_suppress_t_csi = true`

call_init_send_empty_address

Syntax:	`call_init_send_empty_address =` *`true`*`|`*`false`*
Description:	Sends an empty connection address of c=IN IP4 0.0.0.0 in the initial INVITE to parties A and B in a 3PCC call-initiation scenario.
Type:	Boolean
Optionality:	Optional (default used if not set).
Default:	false
Example:	`call_init_send_empty_address = true`

call_init_use_reinvite

Syntax:	`call_init_use_reinvite =` *`true`*`|`*`false`*
Description:	During call-initiation, send the B-leg SDP to the A-leg in a re-INVITE rather than an ACK.
Type:	Boolean
Optionality:	Optional (default used if not set).
Allowed:	true, false
Default:	false

Notes:	Using the ACK reduces the delay where the A-leg hears nothing, but can cause the A-leg to disconnect in networks where the B-leg is slow to respond.
Example:	`call_init_use_reinvite = false`

p_asserted_identity

Syntax:	`p_asserted_identity =` *`true`*`|`*`false`*
Description:	Add a P-Asserted-Identity to the INVITE messages created by the SCA.
Type:	Boolean
Optionality:	Optional (default used if not set).
Allowed:	true, false
Default:	false
Notes:	The identity will be equal to the From: URI.
Example:	`p_asserted_identity = false`

ESC Configuration

Introduction

The `esc` section in the **sca.config** file must be configured to enable the Oracle SIP Chassis (ESC) to work. An example **sca.config** file showing the available `esc` configuration options is installed by the scaScp package in **/IN/services_packages/SCA/etc/sca.config.all.example**.

The **sca.config** file needs to be present on all SCA SLCs.

Note: All necessary configuration for the ESC is done in the **sca.config** file at installation time by the configuration script.

Example esc configuration

The following is an example of `esc` configuration in **sca.config:**

```
esc = {
    default_domain = "telco.com"
    dns_server = "192.168.25.59"
    dns_check_files = true
    use_tcp = true
    tcp_addr = "0.0.0.0"
    tcp_port = 5060
    use_udp = true
    udp_addr = "0.0.0.0"
    udp_port = 5060
    timers = {
        T1 = 5
        T2 = 40
        T3 = 320
        T4 = 50
        T5 = 2400
        MULTIPLIER = 64
    }

    device_list = ["bge0", "bge1"]

    prack_support = true
    error_response_failover=false
    rfc_2543_support = false
    pollInterval = 10
    persistant_connections=true
    send_100_trying=true
    ocsc_call_flow=false
    txn_id_include_host=true

}
```

Parameters

The ESC accepts the following high level parameters.

`default_domain`

Syntax:	`default_domain = "`*`dom`*`"`
Description:	The default domain for this instance of the SCA.
Type:	String
Optionality:	Mandatory
Allowed:	
Default:	None
Notes:	
Example:	`default_domain = "exampletelco.com"`

`device_list`

Syntax:	`device_list = ["`*`name1`*`","`*`name2`*`","`*`namen..`*`"]`
Description:	List of network devices to monitor for their current state (UP or DOWN). If all devices go DOWN, then all active calls will be canceled. If no devices are specified, then this behavior will be disabled.
Type:	Array
Optionality:	Optional (default used if not set).
Allowed:	Network device names
Default:	Empty
Notes:	
Example:	`device_list = ["bge0","bge1"]`

`dns_check_files`

Syntax:	`dns_check_files = `*`true`*`|`*`false`*
Description:	Determines whether or not the SCA should check the **/etc/hosts** file for a DNS entry before performing a DNS lookup.
Type:	Boolean
Optionality:	Optional (default used if not set).
Allowed:	true, false
Default:	false
Notes:	
Example:	`dns_check_files = true`

`dns_server`

Syntax:	`dns_server = "`*`host`*`"`
Description:	The DNS server that the SCA sends queries to.
Type:	String
Optionality:	Optional (default used if not set).
Allowed:	A valid IP address or hostname.
Default:	"127.0.0.1"
Notes:	
Example:	`dns_server = "191.0.2.0"`

`error_response_failover`

Syntax:	`error_response_failover = `*`true`*`|`*`false`*
Description:	Determines whether or not to treat 500 and 503 error responses as a remote address failure and failover to an alternate address.
Type:	Boolean
Optionality:	Optional (default used if not set).
Allowed:	true, false
Default:	false
Notes:	
Example:	`error_response_failover = true`

`persistant_connections`

Syntax:	`persistant_connections = `*`true`*`|`*`false`*
Description:	Sets whether or not to re-use stream connections (such as TCP connections) between transactions. Connections to the same IP:port destination will remain open between transactions until the end point closes the connection.
Type:	Boolean
Optionality:	Optional (default used if not set).
Allowed:	true, false
Default:	true
Notes:	Do not disable this if stream connections will be used in high-traffic environments as the system will quickly run out of sockets.
Example:	`persistant_connections = true`

`pollInterval`

Syntax:	`pollInterval = `*`int`*
Description:	Sets how often to poll the transport layer (in milliseconds).
Type:	Integer
Optionality:	Optional (default used if not set).
Allowed:	A valid integer
Default:	10
Notes:	
Example:	`pollInterval = 20`

`prack_support`

Syntax:	`prack_support = `*`true`*`|`*`false`*
Description:	Defines whether support for PRACK (reliable transmission of provisional responses) is enabled.
Type:	Boolean
Optionality:	Optional
Allowed:	true , false
Default:	false
Notes:	
Example:	`prack_support = false`

`ocsc_call_flow`

Syntax:	`ocsc_call_flow=`*`true`*`|`*`false`*
Description:	If `ocsc_call_flow` is set to `true` then the SCA ignores any SDP sent within a 183-Call-Progress response, thus suppressing any REINVITE that normally would be triggered.
Type:	Boolean
Optionality:	Optional (default used if not set).
Allowed:	true or false
Default:	false
Notes:	Allows the SCA to comply with the OSCS SIP server.
Example:	`ocsc_call_flow=true`

`rfc_2543_support`

Syntax:	`rfc_2543_support = `*`true`*`|`*`false`*
Description:	Defines whether or not support for messages that are not compliant with RFC 3261 is enabled.
Type:	Boolean
Optionality:	Optional (default used if not set).
Allowed:	true, false
Default:	false
Notes:	
Example:	`rfc_2543_support = true`

`send_100_trying`

Syntax:	`send_100_trying = `*`true`*`|`*`false`*
Description:	When set to true, the SCA will send a 100 trying response to an INVITE request to prevent re-transmissions. In addition, 100 trying responses from the B-leg will not be returned.
Type:	Boolean
Optionality:	Optional (default used if not set).
Allowed:	true, false
Default:	true
Notes:	
Example:	`send_100_trying = false`

`tcp_addr`

Syntax:	`tcp_addr = "addr"`
Description:	Defines the address on which to listen for incoming TCP/IP connections.
Type:	String
Optionality:	Optional (default used if not set).
Allowed:	Valid IP address
Default:	"0.0.0.0"
Notes:	
Example:	`tcp_addr = "0.0.0.0"`

`tcp_port`

Syntax:	`tcp_port = port`
Description:	Defines the port on which to listen for incoming TCP/IP connections
Type:	Integer
Optionality:	Optional (default used if not set).
Allowed:	Valid port number
Default:	5060
Notes:	
Example:	`tcp_port = 5060`

`timers`

List of transaction timer parameters within {}.

Default:	-

See *timers section* (on page 82).

`txn_id_include_host`

Syntax: `txn_id_include_host=true|false`

Description: When `txn_id_include_host` is set to *`true`*, the SCA includes the top Via hostname or top Via IP address when generating transaction IDs. Transaction IDs are used to match responses to the original request and are normally generated as follows: *top_via_branch@top_via_hostname*/*CSeq_method*

Where:

- *top_via_branch* is the branch to use from the top Via header.
- *top_via_hostname* is the hostname or IP address to use from the top Via header.
- *CSeq_method* is the Cseq method to use.

However, if the response is received on a different NIC to the one that sent the request, then the IP in the top Via header will be different. This option allows the hostname part of the transaction ID to be turned off and become: *top_via_branch*/*CSeq_method*.

Type:	Boolean
Optionality:	Optional (default used if not set).
Allowed:	true or false
Default:	false
Example:	`txn_id_include_host=true`

`udp_addr`

Syntax:	`udp_addr = "`*`addr`*`"`
Description:	Defines the address on which to receive incoming UDP datagrams.
Type:	String
Optionality:	Optional (default used if not set).
Allowed:	Valid IP address
Default:	"0.0.0.0"
Notes:	
Example:	`udp_addr = "0.0.0.0"`

`udp_port`

Syntax:	`udp_port = `*`port`*
Description:	Defines the port on which to receive incoming UDP datagrams
Type:	Integer
Optionality:	Optional (default used if not set).
Allowed:	Valid port number
Default:	5060
Notes:	
Example:	`udp_port = 5060`

`use_tcp`

Syntax:	`use_tcp = `*`true`*`|`*`false`*
Description:	Determines whether the TCP/IP transport is enabled
Type:	Boolean
Optionality:	Optional (default used if not set).
Allowed:	true, false
Default:	false
Notes:	At least one of `use_tcp` or `use_udp` must be set to true.
Example:	`use_tcp = true`

`use_udp`

Syntax:	`use_udp = `*`true`*`|`*`false`*
Description:	Determines whether the UDP transport is enabled.
Type:	Boolean
Optionality:	Optional (default used if not set).
Allowed:	true, false
Default:	true
Notes:	At least one of `use_tcp` or `use_udp` must be set to true.
Example:	`use_udp = true`

timers section

Here is an example of the `timers` section configuration. Please refer to RFC3261 for more information on timer values.

```
timers = {
    T1 = 5
    T2 = 40
    T3 = 320
    T4 = 50
    T5 = 2400
    MULTIPLIER = 64
}
```

MULTIPLIER

Syntax:	`MULTIPLIER = `*`val`*
Description:	The default transaction timer multiplier. For example a transaction will retransmit a request for 64*T1 (MULTIPLIER*T1) seconds by default.
Type:	Integer
Optionality:	Optional (default used if not set).
Allowed:	A valid number
Default:	64
Notes:	Refer to RFC 3261 for more information.
Example:	`MULTIPLIER = 64`

T1

Syntax:	`T1 = `*`deciseconds`*
Description:	Defines the initial value (in deci-seconds) for the T1 transaction timer.
Type:	Integer
Optionality:	Optional (default used if not set).
Allowed:	A valid integer. The minimum value for T1 is 5.
Default:	5
Notes:	
Example:	`T1 = 5`

T2

Syntax:	`T2 = `*`deciseconds`*
Description:	Defines the initial value (in deci-seconds) for the T2 transaction timer.
Type:	Integer
Optionality:	Optional (default used if not set).
Allowed:	A valid integer
Default:	40
Notes:	
Example:	`T2 = 40`

T3

Syntax:	`T3 = `*`deciseconds`*
Description:	Defines the initial value (in deci-seconds) for the T3 transaction timer.
Type:	Integer
Optionality:	Optional (default used if not set).
Allowed:	A valid integer
Default:	320
Notes:	

Example:	`T3 = 320`

`T4`

Syntax:	`T4 =` *`deciseconds`*
Description:	Defines the initial value (in deci-seconds) for the T4 transaction timer.
Type:	Integer
Optionality:	Optional (default used if not set).
Allowed:	A valid integer
Default:	50
Notes:	
Example:	`T4 = 50`

`T5`

Syntax:	`T5 =` *`deciseconds`*
Description:	Defines the initial value (in deci-seconds) for the T5 transaction timer. The timer sets the value of Timer C (see RFC 3261), which defines the maximum time for which an INVITE transaction can exist without receiving a final response.
Type:	Integer
Optionality:	Optional (default used if not set).
Allowed:	A valid integer
Default:	2400
Notes:	
Example:	`T5 = 2400`

Parser Configuration

Introduction to parser configuration

You can set whether or not the parser is case-sensitive by configuring the `parser` section of the **sca.config** configuration file. For example, the following configuration sets the parser to be case-sensitive:

```
parser = {
    case_sensitive=true
}
```

The parser section of **sca.config** supports the following parameter:

`case_sensitive`

Syntax:	`case_sensitive=`*`true`*`|`*`false`*
Description:	Sets whether or not the parser is case sensitive.
Type:	Boolean
Optionality:	Optional (default used if not set).
Allowed:	true or false
Default:	false
Example:	`case_sensitive=true`

Presence Configuration

Introduction

The `presence` section in the **eserv.config** file defines generic translations for SIP locations and availabilities. It must be configured to enable ACS presenceQuery and presenceSetting chassis actions to work.

The presenceQuery and presenceSetting chassis actions return location and availability information for a subscriber/presence entity from a third party presence server.

This section is for information only, all necessary presence configuration is done at installation time by the configuration script.

The **eserv.config** configuration file is located in **/IN/service_packages/eserv.config**.

Example

Here is an example of the configuration in the `presence` section of **eserv.config**:

Usage:

```
presence = {
    sca_if_handle = "sca"

    pidf_av = {
        open = "Available"
        closed = "N/A"
        away = "Away"
        busy = "Busy"
        default = "N/A"
    }

    pidf_loc = {
        home = "Home"
        office = "Office"
        default = "PlaceOther"
    }

    pidf_rev_av = {
        Available = "open"
        default = "closed"
    }

    pidf_rev_loc = {
        Office = "office"
        default = "home"
    }
}
```

Parameters

The presence server supports the following parameters from the `presence` section of **eserv.config**.

pidf_av

Syntax:	`pidf_av = {` `open = "Available"` `closed = "N/A"` `away = "Away"` `busy = "Busy"` `default = "N/A"` `}`
Description:	Configures the translations from SIP availability definitions to generic availabilities. It is used for both presence querying and presence setting and is dependent on the characteristics of the presence server being used.
Type:	Parameter group
Optionality:	Mandatory
Allowed:	
Default:	
Notes:	
Example:	

pidf_loc

Syntax:	`pidf_loc = {` `home = "Home"` `office = "Office"` `default = "PlaceOther"` `}`
Description:	Configures the translations from SIP location definitions to generic locations. It is used for presence querying and presence setting and is dependent on the characteristics of the presence server being used.
Type:	Parameter group
Optionality:	Mandatory
Allowed:	
Default:	
Notes:	
Example:	

pidf_rev_av

Syntax:	`pidf_rev_av = {` `Available = "open"` `default= "closed"` `}`
Description:	Configures the translations from generic availabilities to SIP availabilities. It is used for both presence setting and presence querying and is dependent on the characteristics of the presence server being used.
Type:	Parameter group
Optionality:	Mandatory
Allowed:	
Default:	
Notes:	
Example:	

pidf_rev_loc

Syntax:	`pidf_rev_loc = {`
	`    Office = "office"`
	`    default = "home"`
	`    }`

Description:	Configures the translations from generic locations to SIP locations. It is used for both presence setting and presence querying and is dependent on the characteristics of the presence server being used.
Type:	Parameter group
Optionality:	Mandatory
Allowed:	
Default:	
Notes:	
Example:	

sca_if_handle

Syntax:	`sca_if_handle = "`*`handle`*`"`
Description:	The SLEE interface handle for the sca.
Type:	String
Optionality:	Mandatory
Allowed:	
Default:	"sca"
Notes:	
Example:	`sca_if_handle = "sca"`

acs.conf Configuration

Introduction

The **acs.conf** file must be configured to enable presence querying and presence setting chassis actions to work.

This section is for information only, all necessary **acs.conf** configuration is done at installation time by the configuration script.

The **acs.conf** configuration file is located in **/IN/service_packages/ACS/etc/acs.conf**.

Refer to *Advanced Control Services Technical Guide* for details on ACS configuration.

acsChassis configuration

On installation the following line is added to the `acsChassis` section of **acs.conf**:

```
chassisPlugin libpresenceSipActions.so
```

SLEE.cfg Configuration

Introduction

The **SLEE.cfg** file must be configured to enable the SCA to work. If you install the Convergent Charging Controller SLEE registrar, then this must also be configured in **SLEE.cfg**.

This section is for information only, all necessary SLEE configuration is done at installation time by the configuration script. Refer to *Service Logic Execution Environment Technical Guide* for details on SLEE configuration.

sca SLEE configuration

On installation the following line is added to the **SLEE.cfg**:

```
INTERFACE = sca sca.sh /IN/service_packages/SCA/bin EVENT
```

Warning: It is essential for the correct operation of the SCA that the SLEE Interface type is always set to EVENT.

registrar SLEE configuration

If you installed the SLEE registrar during the scaScp package installation, then the following lines are added to the **SLEE.cfg**:

```
APPLICATION=registrar registrar.sh /IN/service_packages/SCA/bin 1 1 1000
SERVICE=registrar 1 registrar registrar
SERVICEKEY=INTEGER 200 registrar
```

Note: You specify the service key value, in this case 200, during the installation process.

Configuring EDR Collection

Introduction

The SCA can be configured to produce EDRs for use in post processing as required. All EDR configuration is done in the `sca` section of the **sca.config** file. The EDRs are saved to file in a location specified in the **sca.config**.

EDR collection

EDRs are saved to file in tag/value pairs, separated by "|", in the following form:

```
tag1=value1|tag2=value2
```

They are initially written to the following temporary file:

`SCA-`*`ccyymmddHHMMSS`*`.log`, where:

cc = century
yy = year
mm = month
dd = day
HH = hours
MM = minutes
SS = seconds

This file is closed and moved to its permanent location either when one of the following occurs:

- It contains the maximum number of records as configured in **sca.config**
- It has reached its maximum age (in seconds) as configured **sca.config**

Tip: The locations for the temporary and permanent EDR files are defined in the `cdrTempDir` and `cdrFinalDir` parameters. For details, see *SCA Configuration* (on page 43).

EDR configuration example

EDR collection is enabled by the following lines in the `sca` section of **sca.config**:

```
sca = {
    EnableCDRs = true
    cdrTempDir = "/IN/service_packages/SCA/cdr/open"
    cdrFinalDir = "/IN/service_packages/SCA/cdr/closed"
    cdrSizLimit = 100000
    cdrAgeLimit = 600
}
```

For further information on defining the EDR parameters in **sca.config**, see *SCA Configuration* (on page 43). For further details on the generation and format of EDRs, see *SCA EDRs* (on page 110).

Configuring IN Call Model Triggers

Introduction

This section introduces the configuration requirements of the Convergent Charging Controller IN Call Model. The set of libraries that the IN Call Model provides are used by the SCA to trigger SLEE requests to external IN applications such as ACS.

Note: All necessary configuration for the IN Call Model is done in the **tdp.conf** file at installation time by the configuration script. This section is for information only.

Environment variables

This table describes the UNIX shell environment variables to be configured.

Environment Variable Name	Description	Default Value
TDP_DEFINITIONS	Defines the full path name of the Trigger Detection Point definition file.	/IN/service_packages/SCA/etc/tdp.conf **Note:** The default value will be used if this variable is not set.

Trigger detection point (TDP) definition file

The Trigger Detection Point (TDP) definition file, **tdp.conf**, defines a number of configuration parameters and the trigger tables. These are used to determine when to trigger a call to the external IN application.

If there is no TDP definition file, then a default action is taken where ALL calls are triggered to the external IN application with a service key of 1 (one) and a trigger point of 3 (analyzedInformation), and none of the global configuration parameters are considered set.

Example tdp.conf

The trigger in this example causes all calls to be sent to the external IN application with the service key 1 (one) for ACS, and a trigger point of 3 (analyzed information).

```
# Enable CAMEL3 extensions
CAMEL3
```

```
#Default trigger
3 1 3 request all all
```

Tip: All lines that start with # are treated as comments.

Global configuration parameters

The following configuration parameter may be set once on an individual line in the TDP definition file.

Global Parameter	Description
`CAMEL3`	This parameter enables CAMEL3 extensions. If defined, the called party number is also copied into the intialDP's calledPartyBCDNumber CAMEL3 parameter. The NOA of the called party number becomes the BCD number type.

Trigger detection point definitions

After any global parameters have been set, the configuration file may take one or more trigger detection point (TDP) definitions.

Each line defines a single trigger; its trigger parameter values that get sent and the conditions under which it gets sent.

Each line takes the following form:

```
tdp svcKey eventType msgType cgPn cdPn [wild] [keep]
```

The table below defines the meanings and forms of these parameters.

Global Parameter Value	Type	Description
tdp	integer	This integer value defines the point that the TDP is triggered at. Together with `cgPn`, *cdPn* and *wild* it defines the condition that the trigger will fire on. See the TDP event type table for a list of valid values and meanings.
svcKey	integer	This parameter defines the serviceKey value that will be inserted into the initialDP message when this trigger fires.
eventType	integer	This parameter defines the eventTypeBCSM value that will be inserted into the InitialDP message when this trigger fires. See the TDP event type table for a list of valid values and meanings. Generally this will be the same value as *tdp*.
msgType	request or notify	This parameter defines whether the TDP is sent as a TDP-R (request) or TDP-N(notify). Generally request is used here.
cgPn	*num* or *nat*:*num* or all	This parameter defines the calling party numbers that will trigger the TDP. Together with *tdp*, *cdPn* and *wild* it defines the condition that the trigger will fire on. • *num* defines the prefix of the calling party digits, numbers must begin with these digits for the trigger to fire. • *nat* is optional and defines additionally a nature of address (NOA) of the calling party that must match for

Global Parameter Value	Type	Description
		the trigger to fire. If not provided a nature of 2 (unknown) is assumed. If all is defined then ALL calling party numbers will match.
`cdPn`	*num* or *nat*:*num* or all	This parameter defines the called party numbers that will trigger the TDP. Together with `tdp`, `cgPn` and `wild` it defines the condition that the trigger will fire on. • *num* defines the prefix of the called party digits, numbers must begin with these digits for the trigger to fire. • *nat* is optional and defines additionally a nature of address (NOA) of the called party that must match for the trigger to fire. If not provided a nature of 2 (unknown) is assumed. If all is defined then ALL called party numbers will match.
`wild`	integer	This optional parameter defines the number of digits that must be present in the called party numbers before the TDP will trigger. Together with `tdp`, `cgPn` and `cdPn` it defines the condition that the trigger will fire on. If set the trigger will not fire until the called party number has this number of digits. **Note:** The `wild` parameter can be set to a special value of "stop". If it is set to this value, then the trigger will only fire when a stop digit is received.
`keep`	-	If this optional flag is defined then all numbers triggered by this TDP will keep their stop digits (if they have one).

TDP event type values

The following table defines the list of TDPs as defined by the CS-1 standard. It also defines the point at which the trigger will be instantiated by the Convergent Charging Controller IN Call Model.

TDP	CS-1 Trigger Name	Call Model TDP Creation Point
1	origAttemptAuthorized	digitsReceived
2	collectedInfo	digitsReceived
3	analyzedInformation	digitsReceived
4	routeSelectFailure	released (cause != 16, 17, 18, 19, 21 or 31)
5	oCalledPartyBusy	released (Aparty, cause==17)
6	oNoAnswer	released (Aparty, cause==18, 19 or 21)
7	oAnswer	answered(Aparty)
8	oMidCall	not supported
9	oDisconnect	released (Aparty, cause==16 or 31)
10	oAbandon	released (Aparty, cause==16 or 31)

TDP	CS-1 Trigger Name	Call Model TDP Creation Point
12	termAttemptAuthorized	digitsReceived
13	tCalledPartyBusy	released (Bparty, cause==17)
14	tNoAnswer	released (Bparty, cause==18, 19 or 21)
15	tAnswer	answered(Bparty)
16	tMidCall	not supported
17	tDisconnect	released (Bparty, cause==16 or 31)
18	tAbandon	released (Bparty, cause==16 or 31)
100	n/a	ringing (Aparty)
101	n/a	ringing (Bparty)

Chapter 5

NHP Configuration

Overview

Introduction

This chapter explains how to configure the next hop processing (NHP) for the SCA.

In this chapter

This chapter contains the following topics.

NHP Configuration File 93
NHP Rule Definition 94
NHP Commands 96
NHP Host Definition 99

NHP Configuration File

Introduction

The next hop processing (NHP) configuration defines the rules and associated commands that are used by the SCA to determine how to process SIP messages. You configure the next hop processing that will be performed by the SCA in the NHP configuration file called **rules.nhp**. The NHP configuration file is located in the following directory:

/IN/service_packages/SIP/etc/

The NHP configuration file contains the following sections:

- A RULES section containing the list of rule definitions.
- One or more HOST sections that define the rules to apply to SIP response codes, and the methods (SIP requests) available on the host and the rules to apply to each method.

Example rules.nhp

Here is an example of the NHP rules configuration in **rules.nhp**:

```
RULES = {
    "r1" = ( FROM STARTS "+441473" ) INTERNAL;
    "r2" = ( BODY MATCHES "[0-9].*" ) INTERNAL;
    "r3" = ( TO DOMAIN ENDS "oracle.co.uk" ) FWD ( SK = 1 );
    "r4" = ( TO PORT EQUALS "5959" ) REDIRECT ( URI = "otherhost:5090" );
    "r5" = ( FROM MATCHES "^[0-9].*" AND TO MATCHES "[ABS]" ) IDP ( );
    "r6" = ( TO MATCHES "[0-9].*" OR BODY MATCHES "m=111" ) IDP ( SK = 1 );
    "r7" = ( TO DOMAIN MATCHES "oracle.com" ) LOCATION ( URI = "sip:other2@host2" );
    "r8" = ( DEFAULT ) LOCATION ( SK = 55 );
    "r9" = ( TO DOMAIN NOT STARTS "oracle" AND TO USER NOT ENDS "simon" ) FORWARD (
    );
    "r10" = ( FROM USER STARTS "simon" ) FORWARD ( URI = "host1", "host2" );
    "r11" = ( DEFAULT) LOCATION ( SK = 200 );
    "r12" = ( DEFAULT) PUT_HEADER ( NAME = "route" ID = "ENCODE_URI_ID" );
    "register" = (DEFAULT) FORWARD ( SK = 200 );
    "encode-uri" ( DEFAULT ) GET_HEADER ( NAME = "x-wcs-encode-uri" ID =
    "ENCODE_URI_ID" );
    }

HOST * = {
    METHOD "INVITE" = { "r2" }
    METHOD "OPTIONS" = { "r1" }
    METHOD * = { "r3", "r5", "r6", "r7", "r8", "r9", "r10", "r10", "r10" }
}

HOST "reldevsmp" = {
    STATUSCODE "183" = { "encode-uri" }
    METHOD "REGISTER" = { "registrar" }
    METHOD "INVITE" = { "r11", "r12" }
}

HOST "reldevsmp" = {
    METHOD "OPTIONS" = { "r1" }
    METHOD * = { "r1" }
}
```

NHP Rule Definition

About NHP Rules

NHP rules are defined in the RULES section in the NHP configuration file. Rules have the following format:

```
RULES = {
    "rule_name" = ( condition ) command [flag [flag]] ( parameter = value [parameter
    = value] );
}
```

Where:

- *rule_name* is a unique name for the rule of up to 64 characters.
- *condition* defines the criteria used to trigger the rule command. For more information, see *Specifying Conditions in NHP Rules* (on page 94).
- *command* is the command that is triggered by the rule.
- *flag* is a flag that may be set for the specified command. You many only set flags for specific commands.
- *parameter* is the parameter that is set by the specified command to the specified *value*.

For more information about NHP commands, flags, and parameters, see *NHP Commands* (on page 96).

Specifying Conditions in NHP Rules

You must specify a condition, which must be enclosed in brackets (), for each NHP rule.

A condition comprises either one, or two boolean expressions combined together. The boolean expressions used in conditions have the following format:

Header [*Element*] [NOT] *Operator Constant*

Tip: A boolean expression is a statement that evaluates to true or false.

This table describes the parameters used in boolean expressions in NHP rules.

Parameter	Description
Header	Sets the SIP header field that is used to evaluate the condition. Valid headers are:

Parameter	Description
	• HEADER • ID • FROM • TO • BODY - may only be used with the MATCHES operator • DEFAULT - the DEFAULT header defines the default rule. No other parameters are required and the rule command is triggered unconditionally **Note:** For FROM and TO headers, the characters appearing after the SIP scheme, such as 'sip:', are used. If the header begins with a double quoted string, then the data evaluated will be contained within the '<' and '>' symbols.
`Element`	(Optional) Qualifies the header so that only specific data from the header is evaluated. Valid elements are: • USER – The characters before the '@' symbol. • DOMAIN – The characters after the '@' symbol and, if present, before the ':' (colon). • PORT – The characters after the ':' at the end of the host address. Defaults to 5060 if not present in the header. • VIA – The characters after the '@' symbol.
`Operator`	Defines how the rule is evaluated. Valid operators are: • EQUALS – The data must exactly match the specified *`Constant`*. No regular expression processing takes place. • STARTS – The data must start with the specified *`Constant`*. • ENDS – The data must end with the specified *`Constant`*. • MATCHES – The specified *`Constant`* is interpreted as a regular expression, and the data must match the result. **Tip:** Type NOT in front of the operator to negate it.
`Constant`	The string value (up to 256 characters long) to compare with the data in the header, or header element. The constant value: • Must be enclosed in double quotes ("") • Can contain any number or character that can occur in a SIP address • Can be a regular expression. For more information, see *A note about regular expressions* (on page 96). **Note:** Regular expression constants are pre-compiled when the NHP file is parsed. If a regular expression is invalid and fails to compile, then an error is reported.

Combining boolean expressions

You can combine two boolean expressions in a condition for an NHP rule by using one of the following words:

- AND – The rule command is executed only if both expressions evaluate to true.
- OR – The rule command is executed if either expression evaluates to true.

Examples

```
( FROM MATCHES "^[0-9].*" AND TO MATCHES "[ABC]" )
( TO MATCHES "[0-9].*" OR BODY MATCHES "m=111" )
```

Note: You cannot combine the DEFAULT expression with another expression.

A note about regular expressions

The following is an explanation of regular expressions and their use.

Regular expressions can contain anchors ('^' and '$'), groups ('[ ]') and wild cards ('*'):

- '^' anchors the expression to the start of the string
- '$' anchors the expression to the end of the string
- '[ ]' groups characters into a class, for example [0-9] means any number
- '*' repeats the previous character class zero or more times, such as '[0-9]*'

Examples

This table shows some example expressions.

Expression	Description
^1234$	Will match the exact string 1234.
^1234	Will match any string beginning with 1234.
1234$	Will match any string that ends with 1234.
^[0-9]*1234$	Will match a string that begins with zero or more numbers (0-9) and ends with 1234.
1234[0-9][0-9]*	Will match a string that contains the substring 1234 and then 1 or more numbers (0-9).

NHP Commands

About NHP Commands

When you configure NHP rules you must specify the command that is triggered by each rule. The command defines what to do with the SIP message when the condition for the rule is met. You can specify only one command per rule.

The following commands are supported:

- REDIRECT (or RED)
- FORWARD (or FWD)
- LOCATION (or LOC)
- IDP
- INTERNAL (or INT)
- GET_HEADER
- PUT_HEADER

The NHP commands have the following format:

```
command [flag [flag]] ( parameter = value [parameter = value] )
```

Where:

- *command* is a valid NHP command.
- *flag* is an optional flag that may be set for the specified command.
- *parameter* is the parameter that is set by the specified command to the specified *value*.

Redirect command

The REDIRECT (or RED) command redirects a SIP request to an alternative SIP URI. It accepts one of the following parameters:

- URI = "*SIP_URI*"
- SK = *service_key*

Where:

- `SIP_URI` is the URI to which the request is redirected.
- `service_key` is the service key to which the request is redirected. You must specify an integer.

When a redirect rule is triggered, a 302 Moved Temporarily message is sent in response to the request that triggered the redirect rule. The contact header of this response includes the SIP URI to which the request should be redirected.

If the URI parameter is used, then the contact header includes the SIP URI defined in the rule.

If the SK parameter is used, then an IDP is sent to the defined service key. If the triggered control plan includes a terminate feature node (for example, UATB feature node) and this returns a connect response, then the contact header includes a SIP URI. This is built using the trans_mapping configuration and includes the E.164 number returned in the connect.

Example:

```
Redirect ( URI = "sip:otherhost:5090" )
Redirect ( SK = 1 )
```

Forward command

The FORWARD (or FWD) command forwards SIP requests to one of the following locations:

- An alternative SIP URI
- A SLEE service key
- A destination determined by the Route/Request-URI header if no URI is specified

Flags

The following optional flags may be set by the FORWARD command:

- SET-URI - sets the request URI to the URI specified for the URI parameter. It also appends the user part of the "TO" header from the SIP request to the front of the request URI.
- B2B-UA - this flag will result in the current call being handled as a Back-To-Back User Agent type call.

Example usage:

```
FWD SET-URI,B2B-UA ( URI="sip:192.168.24.178" );
```

Parameters

The FORWARD command accepts one of the following parameters:

- URI = "*SIP_URI*"
- SRV = "*SIP_URI*"
- SK = *service_key*

Where:

- *SIP_URI* is the SIP URI to which the request is forwarded. You can specify a single URI or a comma separated list of up to 100 URIs. The list can be a mixture of IP addresses, and hostnames, that resolve to multiple IP addresses. Each URI must be a complete URI enclosed in double quotes ("").

Example: `"sip:host:port","sip:ipaddress:port","sip:ipaddress"`

If you specify the URI parameter, and *SIP_URI* contains a hostname, then a DNS A record lookup is performed that may return an IP address.

If you specify the SRV parameter, and *SIP_URI* contains a hostname, then a DNS SRV record lookup is performed that may return a list of IP addresses.

- *service_key* is the service key of the SLEE location service.

Note: When forwarding to a list of URIs, the forward command remembers which URI was last used. If the rule is re-triggered then the next URI in the list is used (or the command wraps to the first URI again). This allows the SCA to perform load balancing between URIs when `load_balancing_enabled` (on page 50) is set. The SCA attempts to forward to alternate destinations if an address is unreachable.

Examples:

```
FWD( SK = 55 )
FWD( URI="sip:host1" )
FWD( URI="sip:host1", "sip:host2" )
FWD( SRV="sip:host3")
```

Location command

The LOCATION (or LOC) command performs a location query using an external PAM service.

Flags

The following optional flag may be set by the LOCATION command:

- SET-URI - sets the request URI to the URI returned by the location query.

Example usage:

```
LOCATION SET-URI ( URI = "sip:other2@host2" );
```

Parameters

The LOCATION command accepts one of the following parameters:

- SK = *service_key_of_the_SLEE_based_location_service*
- URI = "*SIP_URI_of_the_location_service*"

Examples:

```
Location ( SK = 55 )
Location ( URI = "sip:other2@host2")
```

IDP command

The IDP command converts the SIP request into a CS1 InitialDP (IDP) and sends it, through the IN-CallModel, to the external IN application. When the IN application responds, the NHP rule evaluation continues from the current position.

Note: This command does not accept any parameters.

Internal command

The INTERNAL (or INT) command uses internal processing rules to respond to the SIP request. It is only used for an OPTIONS request. The SCA responds with the local capabilities in a 200 (OK) response.

Note: This command does not accept any parameters.

PUT_HEADER command

Use the PUT_HEADER command to configure a custom SIP header for inclusion in an outbound SIP INVITE message.

NHP processing applies NHP rules that include the PUT_HEADER command when sending INVITEs during 3PCC call setup.

Configuring PUT_HEADER for Custom SIP Header

The PUT_HEADER command requires a parameter list that consists of a NAME parameter that corresponds to the SIP header name and a VALUE parameter that corresponds to the SIP header value. The following example illustrates the PUT_HEADER command:

```
RULES = { "put" = (DEFAULT)
    PUT_HEADER ( NAME = "x-wcs-cps" VALUE = "late" );
}
HOST * = { METHOD "INVITE" = { "put" }
}
```

Note: PUT_HEADER commands apply *only* to SIP INVITE messages.

GET_HEADER Command

Use the GET_HEADER command to retrieve a header from an inbound SIP response and store the value in memory with a specified *key_name* identifier.

The GET_HEADER command takes the following two parameters:

- NAME = *header_name*
- ID = *key_name*

Note: The GET_HEADER command is allowed only in rules applied to SIP responses.

Example: GET_HEADER

```
RULES = { "r1 = ( DEFAULT ) GET_HEADER (NAME = "x-wcs-encode-uri" ID = "ID_route" );
}
```

NHP Host Definition

About Defining NHP Hosts

You define NHP hosts in one or more HOST sections in the NHP configuration file. Each HOST section defines the rules to apply when processing SIP messages for the named host machine.

You must specify a unique name for each host. The host name is a case sensitive string of up to 64 characters. It must be enclosed in double quotes ("") unless it is the default host. You can define one default host in the NHP file. The default host will be used when a host machine is undefined. It is identified by the host name '*'.

Example: `HOST * = { METHOD "INVITE" = {"R2"}}`

The configuration for each host has the following sections:

- Optional STATUSCODE sections. For information, see *Status Code Rule Definition* (on page 100).
- One or more METHOD sections. For information, see *Defining Host Methods* (on page 100).

Example

```
HOST "HostName" = {
    STATUSCODE "183" = { “encode-uri" }
    METHOD "REGISTER" = { "r1" , "r2" }
    METHOD * = { "r1" }
}
```

Status Code Rule Definition

The STATUSCODE section defines the set of rules to apply to specific SIP responses. It uses the following syntax:

STATUSCODE "*SIP_status*" = { "*RULE*" , "*RULE*" , .. }

Where:

- *SIP_status* is a valid SIP status code for a SIP response, for example, 183.
- Each *RULE* is defined in the RULES section in the **rules.nhp** file. The definitions for the specified rules must contain the GET_HEADER command. For more information about rule definitions, see *NHP Rule Definition* (on page 94).

Defining Host Methods

The METHOD section defines the set of rules to apply to specific SIP requests (methods). You must define at least one method per host. You define methods using the following syntax:

```
METHOD "method_name" = { "RULE" , "RULE" , .. }
```

Where:

- *method_name* is a valid SIP method name, for example, INVITE or REGISTER.
- Each *RULE* is defined in the RULES section in the **rules.nhp** file.

For more information about rule definitions, see *NHP Rule Definition* (on page 94).

Default Method Configuration

The default method has the method name '*'. It defines the default set of rules to apply if the SIP request from the SCA does not match any other method defined for the host. You can define one default method per host.

Example: `METHOD * = { "RULE" }`

Chapter 6

Background Processes

Overview

Introduction

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

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

In this chapter

This chapter contains the following topics.

sca 101

sca

Purpose

The sca process analyzes incoming SIP requests and relays them around the system using different SIP routers.

Startup

The sca process can be run in the following two ways:

1. As a SLEE interface capable of triggering IN applications such as ACS. In this case, the sca is started automatically by the SLEE. For more information, see *SLEE.cfg Configuration* (on page 88).
2. As a standalone binary which cannot trigger IN applications. In this case, the sca can be started from the command line or from inittab.

Example command: This text will start the sca process from the command line:

```
/IN/services_packages/SCA/bin/sca
```

Location

This binary is located on SLCs.

Command line parameters

There are no command line parameters for the sca process.

Configuration

The configuration parameters for the sca process are automatically added to the `sca` section of **sca.config** at installation. For details, see *SCA Configuration* (on page 43).

Failure

If the sca fails, alarms will be raised to the syslog and any incoming SIP messages will not be processed.

Output

The sca process writes output to **/IN/service_packages/SCA/tmp/sca.log**.

Chapter 7

Tools and Utilities

Overview

Introduction

This chapter explains the tools and utilities that are available.

In this chapter

This chapter contains the following topics.

registrar 103
remoteCommanderUser 104
SCA Remote Commander 105
Statistics 108
SCA EDRs 110

registrar

Purpose

The registrar is a SLEE application suitable for registering the IP addresses of SIP contacts. It can be used when processing the following SIP requests:

- Register
- Invite

Register requests

When a SIP REGISTER request is received by the SCA (that is, when a SIP client logs into the system), the SCA sends the IP address of the SIP contact to the registrar, and the registrar stores the address in memory.

Invite request

When the SCA receives an invite request for a specific SIP contact for the first time, it sends a register request to the registrar to retrieve the IP address of the SIP contact. The contact details are then stored in the cache and the invite request is forwarded to the recipient. This means that when the SCA receives subsequent invite requests for the SIP contact, it can retrieve the IP address directly from the cache.

Startup

This process is started automatically by the SLEE, through the shell script **/IN/service_packages/SCA/bin/registrar.sh**.

For details on configuring the SLEE for the registrar application, see *SLEE.cfg Configuration* (on page 88).

Location

This binary is located on SLCs.

Parameters

There are no command line parameters for the registrar application.

Configuration

The registrar is configured by the following parameters. These are automatically added to the `Registrar` section of **sca.config** at installation:

- *cacheSize* (on page 54)
- *defaultExpiry* (on page 54)

Failure

If the registrar fails, then the SCA will not be able to process incoming SIP INVITE and REGISTER requests. Any alarms will be raised to the syslog.

Output

The registrar process writes output to **/IN/service_packages/SCA/tmp/registrar.log**.

remoteCommanderUser

Purpose

The remoteCommanderUser utility sets the password for the SCA Remote Commander and stores the password in a secure credentials vault on the SMS node. The SCA Remote Commander enables remote users to configure and monitor the SCA. See *SCA Remote Commander* (on page 105) for more information.

Startup

You start the remoteCommanderUser utility from the command line by using the following syntax:

```
remoteCommanderUser [-d user/password] [-p RCpassword] [-r]
```

The following table describes the remoteCommander command line parameters.

Parameter	Description
`-d` *`user/password`*	(Optional) The oracle user and password to use to log in to the database. If the `-d` option is not specified then remoteCommanderUser uses the database login specified in the oracleUserAndPassword parameter in the sca section of **eserv.config**. Defaults to '/' if `-d` is not specified and oracleUserAndPassword is not set.
`-p` *`RCpassword`*	(Optional) The new password for the SCA Remote Commander user. remoteCommanderUser Prompts for a password if `-p` is not specified.
`-r`	(Optional) Specifies to delete the password

Setting the SCA Remote Commander Password

Follow these steps to set the password for the SCA Remote Commander.

Step	Action
1	Log in to the SMS as user *smf_oper*.
2	Go to the directory where remoteCommanderUser is located.

3	Enter the following command to set the password for the SCA Remote Commander:
	`remoteCommanderUser -d user/password -p RC_password`
	Where:
	• `user/password` is the login ID for the Oracle database. The login specified in the oracleUserAndPassword parameter is used if you omit the `-d` option. If this is not set, then "/" is used.
	• `RC_password` is the new password for the SCA Remote Commander. remoteCommanderUser prompts for a password if you omit the `-p` option.
	Note: You can remove the SCA Remote Commander password by entering the following command:
	`remoteCommanderUser -r`

SCA Remote Commander

Introduction

The SCA Remote Commander lets you configure and monitor the SCA remotely using a Telnet connection to a specific Remote Commander port.

The SCA Remote Commander is an integral part of the main sca process. It is therefore automatically running whenever the sca process is running. For more information on the sca process and how to start it, see *sca* (on page 101).

About the SCA Remote Commander Password

When you access the SCA Remote Commander, you are prompted to enter the password for the SCA Remote Commander user. You are prompted to set the password for the SCA Remote Commander user when you install the SCA component of Convergent Charging Controller.

You can also set the password for the SCA Remote Commander after you install Convergent Charging Controller by using the remoteCommanderUser utility. See *Setting the SCA Remote Commander Password* (on page 105) for more information.

Configuration

The SCA Remote Commander is configured by the *rem_comm_port* (on page 52) parameter that is automatically added to the **sca.config** file at installation. The rem_comm_port parameter defines the SCA Remote Commander listen port.

Commands

This table describes the SCA Remote Commander commands.

Command	Description
`config`	Starts the SCA Configuration commander which lets you reload any of the available configuration sections.
`diagnostic`	Starts the SCA Diagnostic commander which lets you view and reset the SCA related diagnostic flags.
`statistic`	Starts the SCA Statistic commander which lets you view and modify SCA statistics in real time.
`watcher`	Starts the SCA Watcher commander which lets you monitor SCA CDR, statistic, and diagnostic output, and lets you trace URIs.
`help`	Lists the available commands.
`quit`	Logs you out of the SCA Remote Commander.

Accessing the SCA Remote Commander

Follow these steps to access the SCA Remote Commander.

Step	Action
1	Open a Telnet session on the SCA Remote Commander port. This is the port defined in the `rem_comm_port` parameter in **sca.config**. **Result:** You will be asked for the password. **Note:** If the maximum number of SCA Remote Commander sessions has already been reached, you will be disconnected.
2	Enter the password for the SCA Remote Commander. **Result:** You see this prompt: `SCA Remote Commander>`
3	Enter the command you want, or enter help to display a list of the available commands.
4	To log out of the SCA Remote Commander, enter `quit`.

Configuration commander

Use the SCA Configuration commander to list the available SCA configuration sections and to reload specified sections.

To access the SCA Configuration commander, at the SCA Remote Commander prompt, enter `config`.

This table describes the available commands. At the prompt, enter the command.

Command	Description
`list`	Lists the configuration sections that you can reload.
`reload` *`config_section`*	Reloads the specified configuration section.
`help`	Lists the available configuration commands.
`return`	Returns to the SCA Remote Commander level.
`quit`	Logs out of the SCA Remote Commander.

Diagnostic commander

Use the SCA Diagnostic commander to list and reset the diagnostic flags relevant to the SCA.

To access the SCA Diagnostic commander, at the SCA Remote Commander prompt, enter `diagnostic`.

This table describes the available commands. At the prompt, enter the command.

Command	Description
`list [all]`	Lists the diagnostic flags, and their descriptions, available to the SCA. If you type `list all`, then all registered diagnostic sections are listed without descriptions.
`set` *`flag on|off`*	Turns diagnostic output on or off for the specified flag. Type `set all` *`on|off`* to turn all diagnostic output on or off. **Note:** Flags which are not specific to the SCA can be set.
`get` *`flag`*	Displays the current status of the specified flag.
`save`	Saves the diagnostic flags that are currently set to on.
`load`	Loads the most recently saved diagnostic flags.
`help`	Lists the available diagnostic commands.
`return`	Returns you to the SCA Remote Commander level.
`quit`	Logs you out of the SCA Remote Commander.

Statistic commander

Use the SCA Statistic commander to view and modify SCA statistics in real time.

To access the SCA Statistic commander, at the SCA Remote Commander prompt, enter `statistic`.

This table describes the available commands. At the prompt, enter the command.

Command	Description
`list`	Lists the configuration sections that you can reload.
`set` *`id value`*	Sets the statistic for the specified ID to the value specified.
`delta` *`id value`*	Updates the statistic for the specified ID by the value specified.
`get` *`id`*	Retrieves the current value for the specified statistic ID.
`help`	Lists the available statistic commands.
`return`	Returns you to the SCA Remote Commander level.
`quit`	Logs you out of the SCA Remote Commander.

Watcher commander

Use the SCA Watcher commander to:

- View SCA related EDR, statistic and diagnostic output
- Trace URIs

To access the SCA Watcher commander, at the SCA Remote Commander prompt, enter `watcher`.

This table describes the available commands. At the prompt, enter the command.

Command	Description
`cdr`	Invokes the cdr watcher.
`stats`	Invokes the statistics watcher.
`diagnostic`	Invokes the diagnostic output watcher.
`trace` *`uri`* `[`*`file`*`]`	Traces the specified URI. The output may be redirected to the specified file, if required.
`help`	Lists the available watcher commands.
`return`	Returns you to the SCA Remote Commander level.
`quit`	Logs you out of the SCA Remote Commander.

Statistics

Introduction

The SCA collects statistics using the standard SMS statistic mechanism. The smsStatsDaemon determines which statistics to collect according to a predefined list. If ORACLE is:

- *Installed* on the SLC, then the statistics list is stored in SMF_STATISTICS_DEFN table on the SMS and then replicated to the SLC.
- *Not installed*, then the statistics list is stored in the **/IN/service_packages/SCA/etc/stats_config** file.

The smsStatsDaemon uses replication to update the statistics on the SMS. When starting the smsStatsDaemon, the local node ID is specified through the –r parameter on the command line. It must be in the range 512-1023. If ORACLE is not installed on the SLC, then you must also specify the location of the stats_config file (-f parameter).

The replication configuration file contains the IP address of the SMS.

Example: This starts smsStatsDaemon for an SLC that doesn't have ORACLE installed. The location of the stats_config file is **/IN/service_packages/SCA/etc/stats_config** and the replication node number is 700.

```
/IN/service_packages/SMS/bin/smsStatsDaemon -f
/IN/service_packages/SCA/etc/stats_config -r 700
```

Tip: For more information about the smsStatsDaemon and how statistics are collected, see *Service Management System Technical Guide*.

stats_config file

The **stats_ config** file lists the statistics collected by the smsStatsDaemon for the SCA. It is installed automatically when the scaScp package is installed. Statistics defined in the **stats_config** file have the following format:

```
MID=StatisticID,ApplicationID,Description,Period,[Comment]
```

The available parameters are:

Parameter	Description
StatisticID	The event that occurred.
ApplicationID	The application for which the statistic was generated. This is always SCA.
Description	Describes the statistic.
Period	Determines how frequently (in seconds) the statistic will be output to file.
Comment	Provides any additional comments.

Note: This file is only required where ORACLE is not installed on the SLC.

Example stats_config file

The following is an example of statistics configuration in the **stats_config** file:

```
MID=IN-REQUEST,SCA,Incoming SIP request,300
MID=OUT-REQUEST,SCA,Outgoing SIP request,300
MID=QUERY-IN,SCA,IN trigger (sending InitialDP),300
MID=QUERY-LOC,SCA,Location query,300
MID=CACHE-HIT,SCA,Cache hit,300
MID=CACHE-MISS,SCA,Cache miss,300
MID=IN-INVITE,Incoming INVITE request,300
MID=IN-REGISTER,Incoming REGISTER request,300
MID=IN-MESSAGE,Incoming MESSAGE request,300
MID=IN-CANCEL,Incoming CANCEL request,300
MID=OUT-INVITE,Outgoing INVITE request,300
MID=OUT-REGISTER,Outgoing REGISTER request,300
MID=OUT-MESSAGE,Outgoing MESSAGE request,300
MID=OUT-CANCEL,Outgoing CANCEL request,300
MID=ERR-REQUEST,Request failure (4xx) received,300
MID=ERR-SERVER,Server failure (5xx) received,300
MID=ERR-GLOBAL,Global failure (6xx) received,300
MID=PRL_HUNT,Parallel hunting attempts,300
```

Statistics collected

This table describes the statistics that are collected.

Field	This statistic is incremented each time...
IN-REQUEST	There is a new incoming SIP request.
OUT-REQUEST	There is a new outgoing SIP request.
QUERY-IN	An IN trigger (sending InitialDP) is sent.
QUERY-LOC	There is a new location query.
CACHE-HIT	The Cache is hit.
CACHE-MISS	There is a Cache miss.
IN-INVITE	There is an incoming INVITE request.
IN-REGISTER	There is an incoming REGISTER request.
IN-MESSAGE	There is aa incoming MESSAGE request.
IN-CANCEL	here is an incoming CANCEL request.
OUT-INVITE	There is an outgoing INVITE request.
OUT-REGISTER	There is an outgoing REGISTER request.
OUT-MESSAGE	There is an outgoing MESSAGE request.
OUT-CANCEL	There is an outgoing CANCEL request.
ERR-REQUEST	A request failure (4xx) is received.
ERR-SERVER	A server failure (5xx) is received.
ERR-GLOBAL	A global failure (6xx) is received.
PRL-HUNT	Parallel hunting is attempted.

SCA EDRs

EDR collection

The SCA can be configured to produce EDRs for use in post processing as required. The EDRs are saved to file in a location specified in the **sca.config**.

EDRs are saved to file in tag/value pairs, separated by "|", in the following form:

```
tag1=value1|tag2=value2
```

Field formats

Each field in an EDR is in a particular format, summarized in this table.

Format	Description
Date / Time	A time to the nearest second, in format *YYYYMMDDHHmmSS* where: • *YYYY* = year (for example, 2005) • *MM* = month (for example, 03 for March) • *DD* = day of the month (for example, 09) • *HH* = hours (for example, 13 for 1pm) • *mm* = minutes (for example, 32) • *SS* = seconds (for example, 00) **Example:** A request submitted on 16th November 2007 1 minute and 14 seconds after midnight `TIMESTAMP=20071116000114`
Integer	A decimal number. Will never exceed a 32 bit number (11 digits), but is often shorter. Leading zeros will not normally be present. **Example:** `DURATION=30`
String	String of characters. Can be any length. Should not contain the characters = or **|**. May include spaces. When the parameter is a string, the string consists of all the characters after the = sign up to the **|** separator between this parameter and the next. **Example:** `REQUEST_URI=aname@oracle.com;SLEESK=1`

Notes: Tags may not necessarily be in a fixed order, as the order of processing may vary from one call type to another.

EDR fields

Here are the SCA tags within an EDR.

CDR_TYPE (sca reason for record generation)

Description:	Type of EDR (that is, where and why it was generated).
Format:	Integer
Version:	SCA 1.0
Notes:	1 - Call attempt 2 - Success 3 - Error
Example:	`CDR_TYPE=2`

`DURATION (session duration)`

Description:	The session duration (in seconds).
Format:	Integer
Version:	SCA 1.0
Notes:	This tag value is only present where the CDR_TYPE is 2.
Example:	`DURATION=30`

`FROM (sip message from header)`

Description:	Contains the contents of the From header in the SIP message.
Format:	String
Version:	SCA 1.0
Notes:	
Example:	`FROM=`

`METHOD (sip method of request)`

Description:	The SIP method for the request that caused the EDR to be generated.
Format:	String
Version:	SCA 1.0
Notes:	
Example:	`METHOD=`

`REQUEST_URI (uri request content)`

Description:	Contains the contents of the URI request.
Format:	String
Version:	
Notes:	
Example:	`REQUEST_URI=aname@oracle.com;SLEESK=1`

`TIMESTAMP (creation timestamp of sca edr)`

Description:	The date and time when the EDR was generated.
Format:	Date
Version:	SCA 1.0
Notes:	
Example:	`TIMESTAMP=20071116000114`

`TO (sip to header content)`

Description:	Contains the contents of the To header in the SIP message.
Format:	String
Version:	
Notes:	
Example:	`TO=`

Chapter 8

About Installation and Removal

Overview

Introduction

This chapter provides information about the installed components for the Oracle Communications Convergent Charging Controller application described in this guide. It also lists the files installed by the application that you can check for, to ensure that the application installed successfully.

In this Chapter

This chapter contains the following topics.

Installation and Removal Overview 113
Checking the Installation 113

Installation and Removal Overview

Introduction

For information about the following requirements and tasks, see *Installation Guide*:

- Convergent Charging Controller system requirements
- Pre-installation tasks
- Installing and removing Convergent Charging Controller packages

SCA packages

An installation of Session Control Agent includes the following packages, on the:

- SMS:
  - scaSms
- SLC:
  - scaScp

Checking the Installation

Introduction

Refer to these check lists to ensure the Session Control Agent has been installed correctly.

SCA directories and files

The SCA installation creates the following directories:

- **/IN/service_packages/SCA/bin**
- **/IN/service_packages/SCA/cdr**

- **/IN/service_packages/SCA/etc**
- **/IN/service_packages/SCA/lib**
- **/IN/service_packages/SCA/tmp**

The SCA installation installs the following binaries and interfaces:

- **/IN/services_packages/SCA/bin/sca**
- **/IN/services_packages/SCA/bin/registrar**

The SCA installation installs the following example configuration files:

- **/IN/services_packages/SCA/etc/rules.nhp**
- **/IN/services_packages/SCA/etc/sca.config**
- **/IN/services_packages/SCA/etc/sca.config.all.example**
- **/IN/services_packages/SCA/etc/stats_config**
- **/IN/services_packages/SCA/etc/tdp.conf**