Oracle® Communications Network Charging and Control

Diameter Control Agent Technical Guide

Release 12.0.0

December 2017

ORACLE®

Copyright

Contents

About This Document

Scope

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

Audience

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

A familiarity with the Diameter protocol is also required. Refer to the following:

- Internet Engineering Task Force (IETF) specifications:
  - RFC 3588 – Diameter Base Protocol
  - RFC 4006 – Diameter Credit-Control Application
  - RFC 4005 – Diameter Network Access Server Application
- 3GPP TS 32.299 V11.3.0 (2012-03) – 3rd Generation Partnership Project; Technical Specification Group Service and System Aspects; Telecommunication management; Charging management; Diameter charging applications (Release 11)

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*
- *Charging Control Services Technical Guide*
- *Charging Control Services 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 Network Charging and Control (NCC) documentation.

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

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

Chapter 1

System Overview

Overview

Introduction

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

It is not intended to advise on any specific Oracle Communications Network Charging and Control (NCC) network or service implications of the product.

In this Chapter

This chapter contains the following topics.

What is Diameter Control Agent?

Introduction

The Diameter Control Agent (DCA) is a SLEE interface used to translate between Diameter messages and CAP 3 INAP.

Diameter is a protocol which has been designed to supersede RADIUS, and which facilitates AAA (Authentication, Authorization and Accounting), and Credit-Control. This protocol forms the basis of a Credit-Control solution for Oracle IMS (IP Multimedia Subsystem) products.

The DCA acts as a Diameter based credit control server. In doing so, it provides an interface to the Prepaid Charging product (CCS component), to facilitate the use of the billing functionality provided there.

Features

The DCA provides the following features:

- Provides support for AVPs specified in TS 32.299.
  - Allows an AVP to be mapped to any INAP operation argument; for example, InitialDP.calledPartyNumber
  - Allows AVPs from any CCR to be mapped to ACS profile fields
  - Allows ACS Profile fields to be mapped to any CCA response sent to a CC-Client
  - Allows specification of complex mappings between AVPs and ACS Profile fields (including type ARRAY)
- Supports call-screening without the need to start a billing session.
- Provides free call support (the ability to send DIAMETER_CREDIT_CONTROL_NOT_APPLICABLE as a response).
- Support for Mobile Network operators (MNO) who provide their own variations on 3GPP and IETF Diameter standards for Credit Control, including:
  - Service triggering

- Extensions to standards based enumerated values
- Default units may be assumed by operators, but be specified explicitly later in the call flows
- Ability to track elapsed-time at the interface and report back to the client
- Allows result-codes set by the interface to be mapped
- Supports the use of non-standard capabilities negotiation

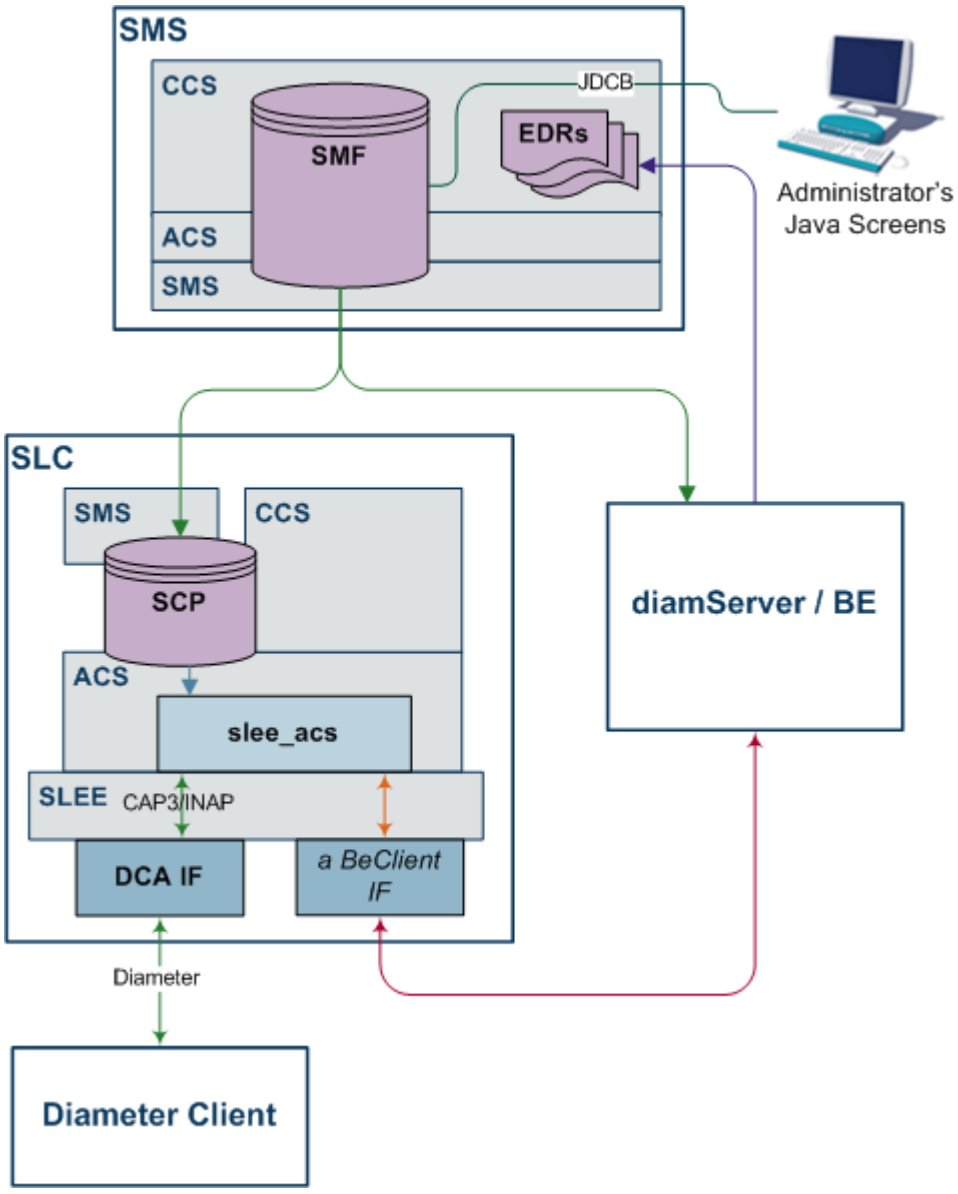
Per-Message Type AVP Mapping

The AVP mapping rules configurable in DCA are:

- CCR INITIAL_REQUEST
- CCR EVENT_REQUEST
- CCA INITIAL_REQUEST
- CCA EVENT_REQUEST
- CCR UPDATE_REQUEST (one or more existing services) + CCA UPDATE_REQUEST
- CCR TERMINATION_REQUEST + CCA TERMINATION_REQUEST

Diagram

Here is a high level diagram showing Diameter Control Agent in the context of NCC components.



DCA Components

In this diagram, the components that are specific to DCA are:

- Diameter Client
- Diameter messages
- DCA interface
- CAP3/INAP within the SLEE

Screening

DCA screening provides the ability to configure one or more service entries in the **eserv.config** file that do not specify a Service-Identifier or a Rating-Group. Additionally, these services contain a flag indicating that they are for screening. Screening is only available if no default Rating-Group has been specified in the config file.

When a Diameter Initial CCR is received by DCA with no Service-Identifier or Rating-Group, DCA will check the configured services for a service that matches the remaining AVPs. If such a service is found, then the corresponding control plan will be triggered.

The control plans used for screening can perform logic, and will return the result of screening by returning either a continue for success, or a release for failure. In both cases, extra information can be passed between the CCR/CCA and ACS using the inbound and outbound AVP mapping features listed in *Per-Message Type AVP Mapping* (on page 2).

On receipt of the continue or release from ACS, DCA will shut down the dialog to ACS, and return the relevant response to Diameter.

Refer to the following:

- Configuration file parameter *screeningService* (on page 67) in the Services section
- ACS Control Plan for *Screening* (on page 130)

Chapter 2

Configuration

Overview

Introduction

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

In this chapter

This chapter contains the following topics.

Configuration Overview

Introduction

This topic provides a high level overview of how the Diameter Control Agent (DCA) interface is configured.

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

Configuration Components

The Diameter Control Agent is configured by the following components:

Component	Locations	Description	Further Information
eserv.config	All SLC machines	DCA is configured by the `DIAMETER` section of **eserv.config**.	**eserv.config** *Configuration* (on page 6).
eserv.config	All SLC machines	DCA services mappings are configured in the `ccsServiceLibrary` section of **eserv.config**.	*CCS Service Library configuration* (on page 10).
SLEE.cfg	All SLC machines	The SLEE interface is configured to include the DCA	*SLEE.cfg Configuration* (on page 7) and *SLEE Technical*

Component	Locations	Description	Further Information
		service.	*Guide.*
acs.conf	All SLC machines	Configures the cc extension Digits/INAP extension mappings	*acs.conf Configuration* (on page 9)

eserv.config Configuration

Introduction

The **eserv.config** file is a shared configuration file, from which many Oracle Communications Network Charging and Control (NCC) applications read their configuration. Each NCC machine (SMS, SLC, and VWS) has its own version of this configuration file, containing configuration relevant to that machine. The **eserv.config** file contains different sections; each application reads the sections of the file that contains data relevant to it.

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

The **eserv.config** file format uses hierarchical groupings, and most applications make use of this to divide the options into logical groupings.

Configuration File Format

To organize the configuration data within the **eserv.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 can be used, as in this example:

```
{ 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" ]
}
```

eserv.config Files Delivered

Most applications come with an example **eserv.config** configuration in a file called **eserv.config.example** in the root of the application directory. The example file for DCA is:

/IN/service_packages/eserv.config.dca.example

Warning: This file is not intended to be changed by the User. Please contact the Oracle support with your queries.

Editing the File

Open the configuration file on your system using a standard text editor. Do not use text editors, such as Microsoft Word, that attach control characters. These can be, for example, Microsoft DOS or Windows line termination characters (for example, ^M), which are not visible to the user, at the end of each row. This causes file errors when the application tries to read the configuration file.

Always keep a backup of your file before making any changes to it. This ensures you have a working copy to which you can return.

Loading eserv.config Changes

If you change the configuration file, you must restart the appropriate parts of the service to enable the new options to take effect.

Diameter eserv.config Configuration

The **eserv.config** file must be configured to enable the DCA to work. All necessary DCA configuration in **eserv.config** is done at installation time by the configuration script. However, you must manually map the service handles for services to the libdcaCcsSvcExtra.so library in the ccsPluginExtend section of ccsServiceLibrary. Refer to *CCS Service Library configuration* (on page 10).

Note: The DCA configuration options in **eserv.config** are explained in the section on the *diameterControlAgent background process* (on page 26).

SLEE.cfg Configuration

Introduction

The **SLEE.cfg** file must be configured to enable the DCA to work. All necessary SLEE configuration is done at installation time by the configuration script, this section is for information only.

The SLEE configuration file is located at **/IN/service_packages/SLEE/etc/SLEE.cfg**.

For more information on SLEE configuration, see *SLEE Technical Guide*.

DCA SLEE Configuration

On installation, the following lines are added to the **SLEE.cfg** configuration file.

```
INTERFACE=dcaIf dca.sh /IN/service_packages/DCA/bin EVENT
SERVICEKEY=INTEGER 1230 Dca_Session
SERVICEKEY=INTEGER 1231 Dca_DD
SERVICEKEY=INTEGER 1232 Dca_RA
SERVICEKEY=INTEGER 1233 Dca_CB
SERVICEKEY=INTEGER 1234 Dca_PE
SERVICE=Dca_Session 1 slee_acs Dca_Session
SERVICE=Dca_DD 1 slee_acs Dca_DD
SERVICE=Dca_RA 1 slee_acs Dca_RA
```

```
SERVICE=Dca_CB 1 slee_acs Dca_CB
SERVICE=Dca_PE 1 slee_acs Dca_PE
```

Note: It is essential for the correct operation of this application that the SLEE Interface type is always set to EVENT.

SERVICEKEY

The SERVICEKEY entries specify the SLEE service keys for the Diameter service.

During dcaScp installation the value of the service keys can be specified, or modified manually after installation, if necessary.

RAR Configuration

Diameter Charging Driver (DCD) can forward re-authorization requests (RARs) to DCA for forwarding on to network elements such as the Online Charging Server (OCS). You enable DCA to process RARs by setting the following parameter:

```
rarHandlingEnabled = true
```

When RAR processing is enabled in DCA, DCA informs DCD that it can process RAR messages, and DCD uses this information to pass on any RARs it receives from the OCS. DCA then forwards the re-authorization response that it receives back from the network to DCD, for DCD to forward on to the OCS. If DCD is not informed by DCA, then DCD will respond to a RAR with a re-authorization acknowledgement (RAA) containing the corresponding result code set in the `rarResultCode` parameter.

Add the following parameters to the DIAMETER configuration section in the **eserv.config** file to enable DCA to process RARs:

```
DIAMETER = {
DCAInstances = {

rarHandlingEnabled = true
rarClientTimeout = int
rarMaxRetry = int

}
}
```

See *DCAInstances Parameters* (on page 40) for more information.

You must also add service library entries to the CCS section of the **eserv.config** file using the following syntax:

```
CCS = {
ccsServiceLibrary = {

ccsPluginExtend = {
library="libdcdCcsSvcExtra.so"
handleName="Dca_Session"
}
}
}
```

where `Dca_Session` is the name of the DCA session service.

Ensure the corresponding SERVICEKEY and SERVICE entries in the SLEE configuration file (**SLEE.cfg**) are present for the DCA session service. For example:

```
SERVICEKEY=INTEGER 1230 Dca_Session
SERVICE=Dca_Session 1 slee_acs Dca_Session
```

Add the following parameters to the CCS configuration section in the eserv.config file on VWS to enable DAP IF to handle out-of-band balance update notification events, such as VWS balance top-ups or depletions, which are likely to affect obtainable reservation durations. In these scenarios, the client (by means of a server-initiated RAR message) is prompted to resubmit a new CCR-UPDATE:

```
CCS = {
    # dcaResPlugin.so config
    dcaResPlugin = {
        # Operation set for RAR notifications
        dapOperationSet = "RAR"
    }
}
BE = {
 plugins = [
     # Final plugin:
     "dcaResPlugin.so"
 ]
}
```

For more information on RAR processing in DCD, see the RAR configuration section in *Diameter Charging Driver Technical Guide*.

acs.conf Configuration

Introduction

The **acs.conf** file must be configured to enable the application to work. All necessary configuration is done at installation time by the configuration script; this section is for information only.

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

Refer to *ACS Technical Guide* for details on ACS configuration.

INAP Extension

The following values for cc extension Digits and INAP extension mappings and format are set in **acs.conf** on installation. You can change these, if required.

```
extensionNumber 3 506 asn1Integer value
extensionNumber 4 507 asn1Integer value
extensionNumber 5 501 asn1Integer value
extensionNumber 6 502 asn1Integer value
extensionNumber 7 503 asn1Integer value
extensionNumber 8 504 asn1Integer value
extensionNumber 9 505 octets value
```

Note: The `extensionNumber n` is displayed as `CC Extension Digits n` in the drop-down fields (for example, **Number of Events**) in the macro node configuration screens. See *Control Plans* (on page 123) for examples.

Prepaid Charging Configuration

CCS Service Library configuration

In order for the PRICE_ENQUIRY, DIRECT_DEBITING and REFUND_ACCOUNT services to work properly, you must manually map the service handles for these services to the libdcaCcsSvcExtra.so library in the `CCS.ccsServiceLibrary.ccsPluginExtend` section of the **/IN/service_packages/eserv.config** file. For example:

```
CCS = {

    ...

    ccsServiceLibrary = {

    ...

        ccsPluginExtend = [
        {
            library="libdcaCcsSvcExtra.so"
            handleName="Dca_PE"
        }
        {
            library="libdcaCcsSvcExtra.so"
            handleName="Dca_DD"
        }
        {
            library="libdcaCcsSvcExtra.so"
            handleName="Dca_RA"
        }
    ]

    ...
```

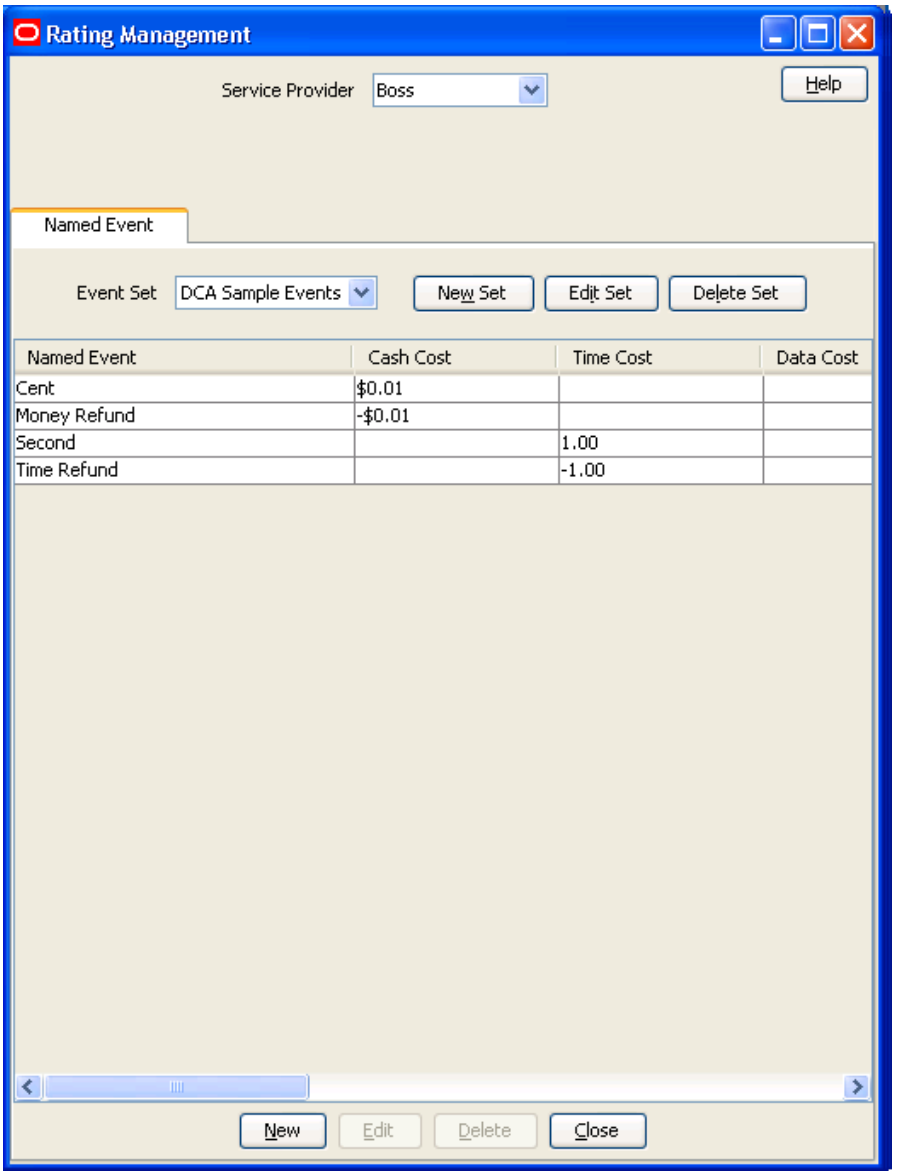
Enabling Named Events

The DCA installation does as much as possible to be usable as soon as it is installed. However, you need to perform one manual procedure in Prepaid Charging before you try to use it first-off. You need to allow the use of the named events that are installed to whatever product types that you employ,

Here is an example of the procedure to follow to allocate product types to a DCA event set.

Step	Action
1	In the SMS main screen, open **Services** > **Prepaid Charging** > **Rating Management**. **Result:** The Rating Management screen will display.
2	Select the **Named Event** tab.
3	From the **Event Set** drop-down box, select `DCA Sample Events`.

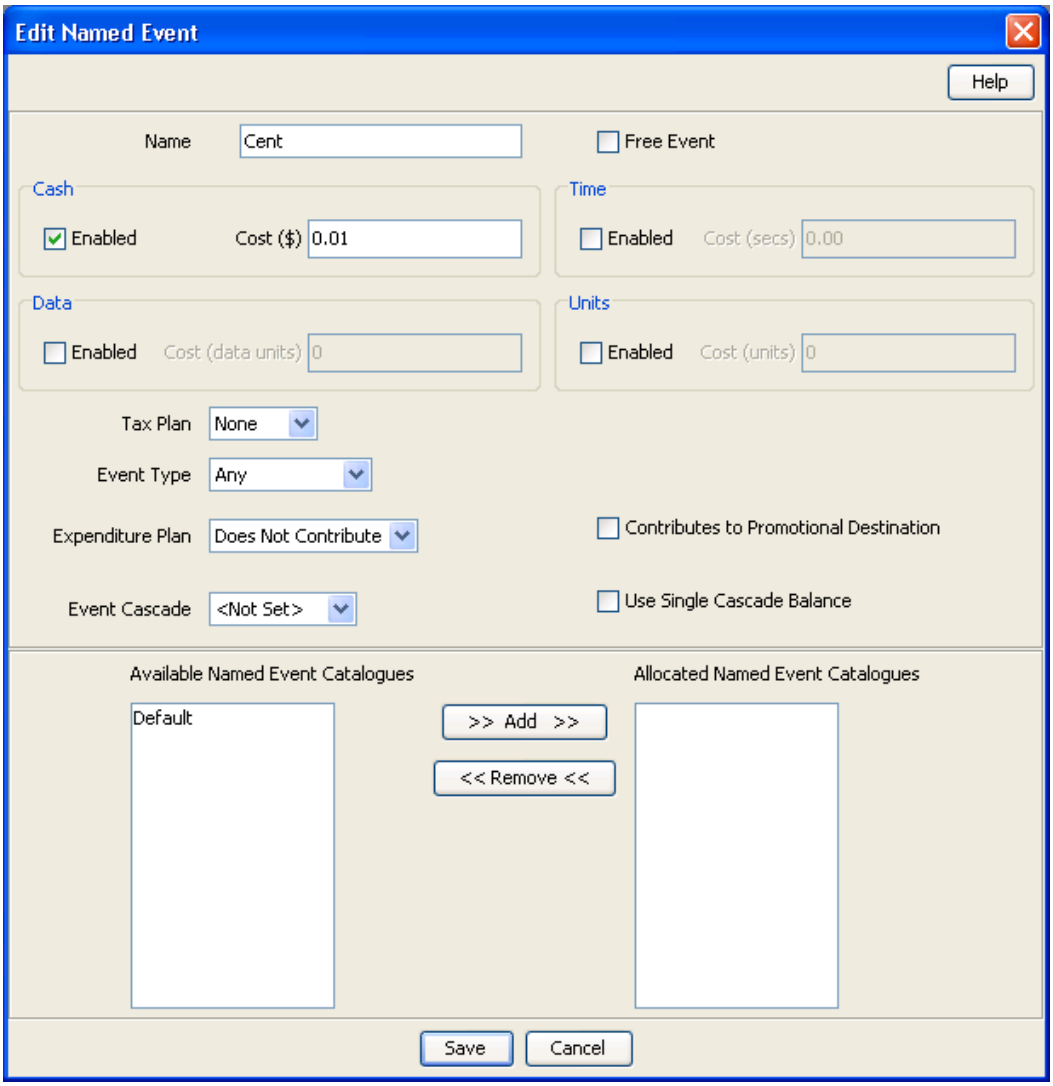
Step	Action



4 From the Named Event grid, select `Cent` and click **Edit**.

Result: The Edit Named Event screen appears.

Step	Action



5 From the **Available Named Event Catalogues** field, select the named event catalogues for this event that will use DCA billing and click **>> Add >>**.

6 Click **Save**.

7 Repeat steps 4 through 6 for the `Second` named event.

Refer to *CCS User's Guide, Named Event* topic for details.

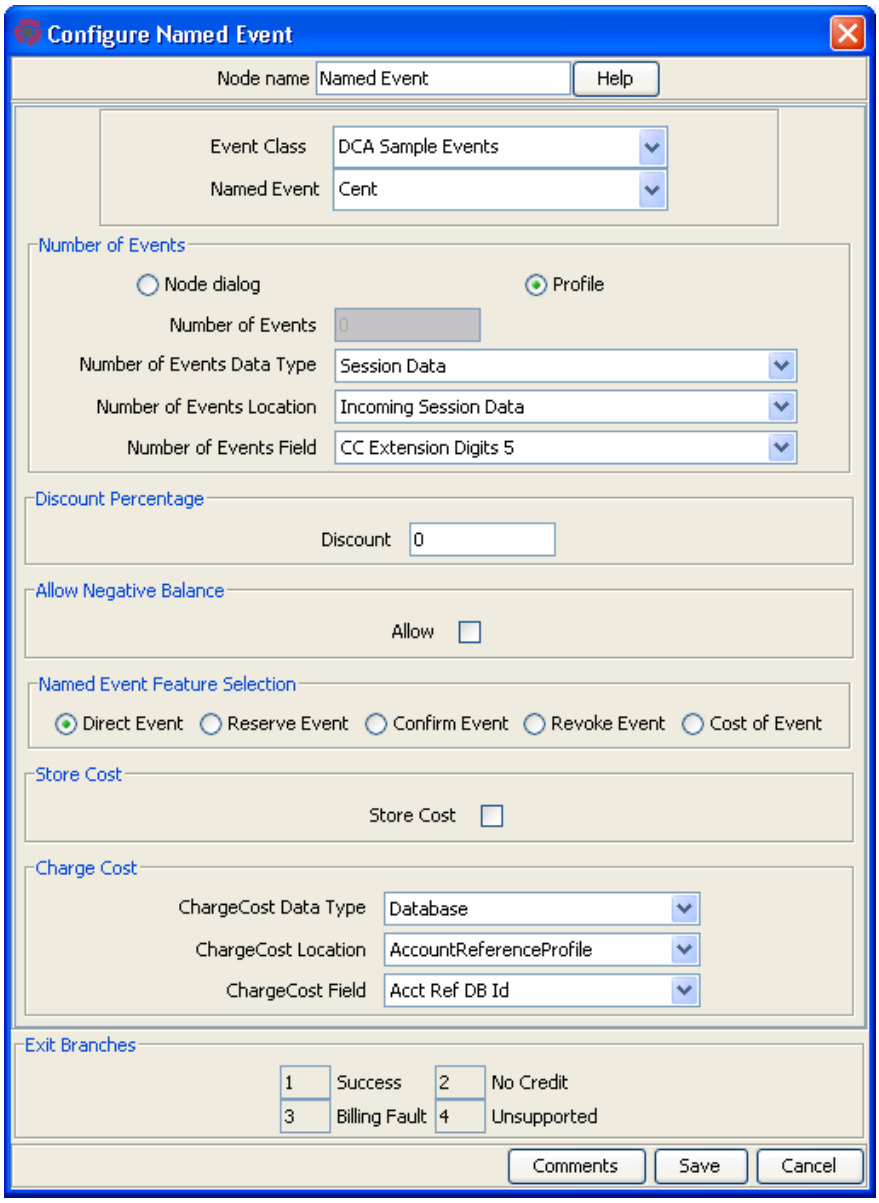
Feature Node Configuration

Named Event Node

The Named Event node must be configured as shown for the following fields:

- Event Class – A Diameter (DCA) event class
- Number of Events Location – `Incoming Session Data`

- Number of Events Field – `CC Extension Digits 5`



INAP Extensions

Introduction

As INAP is not designed to contain Diameter AVPs, these will be carried, where necessary, in INAP extensions in the InitialDP or the Connect. The following pre-defined INAP extension types are used, where appropriate.

The IDP extensions are used by the service loader plug-in to modify the CCS/ACS call context. Also, the control plans may access these extensions by means of suitable **acs.conf** configuration and by use of the ExtensionDigits[0-9] call context fields. See *Control Plans* (on page 123) for examples of control plans using these extensions.

In addition, inbound extension profiles may be set using the `encodedExtension` and `extensionFormat` parameters. This enables inbound AVPs within INITIAL or EVENT based Credit-Control-Request messages to be identified for mapping into the IDP passed to ACS. Multiple AVPs can be identified and passed to the target profile tags available within the inbound extensions block.

Note: While you can have multiple AVP mappings, you can have only *three* extension mappings from DCA to slee_acs. You can create extension mappings either by specifying an encoded extension value, for example extensionType = 508, or by encoding as an extension profile block, which is extension type 701. Note that all profile tags go into one profile block and therefore use only one extension.

Therefore, if you define a profile encoded AVP, you have only 2 more user-defined extensions available. For example, you can have either three AVPs mapped directly to INAP extensions or two AVPs mapped directly to INAP extensions and multiple AVPs that are encoded in one profile block that is mapped to extension type 701.

IDP

The following standard INAP extensions are used in the IDP. This table also lists the mapping of the INAP extensions to the Call Content extension Digits profile buffers.

Extension	Description	Type	cc extension Digits
501	Requested-Service-Units	Asn1Integer	5
502	Requested service unit type: • 1 = CC-Time • 2 = CC-Money • 3 = CC-Total-Octets • 4 = CC-Input-Octets • 5 = CC-Output-Octets • 6 = CC-Service-Specific-Units	Asn1Integer	6
503	Requested-action: • 0 = DIRECT_DEBITING • 1 = REFUND_ACCOUNT • 2 = CHECK_BALANCE • 3 = PRICE_ENQUIRY	Asn1Integer	7
504	Event-Timestamp	Asn1Integer	8
505	Subscription ID	Asn1OctetString	9
506	Currency	Asn1Integer. Value from ISO 4217, for example, 978 = Euro	3
507	Exponent	Asn1Integer. Currency exponent + 0x20. for example, 1E for -2	4
701	Multiple encoded AVPs	Inbound extension profile block	

Note: The cc extension Digits-INAP extension mappings are set in **acs.conf** on installation and can be changed, if required. See *acs.conf Configuration* (on page 9).

Connection

The following INAP extensions are used in the Connect operation.

Extension	Description	Type
601	Granted service units	Asn1Integer
602	Granted service unit type: • 1 = CC-Time • 2 = CC-Money • 3 = CC-Total-Octets • 4 = CC-Input-Octets • 5 = CC-Output-Octets • 6 = CC-Service-Specific-Units	Asn1Integer
603	Cost information (in system currency)	Asn1OctetString

Parameter Mappings

Introduction

This topic describes the mappings between INAP parameters and Diameter AVPs.

CCR

This table describes the mappings for Credit-Control-Request AVPs.

AVP	Action
`Session-Id`	Used to look up the correct StateMachine in **sessionIdToStatemachine**.
`Origin-Host`	The stack code in the DIAMETER module handles this.
`Origin-Realm`	The stack code in the DIAMETER module handles this.
`Destination-Realm`	The stack code in the DIAMETER module handles this.
`Auth-Application-Id`	Throw it out if not 4
`Service-Context-Id`	Used as part of the key to look up the service.
`CC-Request-Type`	Used as part of the key to look up the service. Also used to determine the next state in the state machine.
`CC-Request-Number`	Used in duplicate detection.
`Destination-Host`	The stack code in the DIAMETER module handles this.
`User-Name`	Ignored unless mapped to an IDP extension by the AVP mappings in **eserv.config**.
`CC-Sub-Session-Id`	Ignored after copying from the request to the answer message. We do not support multiple session IDs but some clients may set this anyway so we just ignore it.
`Acct-Multi-Session-Id`	Ignored after copying from the request to the answer message. We do not support multiple session IDs but some clients may set this anyway so we just ignore it.
`Origin-State-Id`	Used to detect a client re-booting and wipe sessions for the host if it has

AVP	Action
	rebooted.
Event-Timestamp	For EVENT_REQUEST messages, this gets copied into IDP extension type 504.
Subscription-Id	One or more Subscription-Id AVPs may be supplied. The first SIP or E164 type Subscription-Id is copied to: • CallingPartyNumber after applying the configured normalization rules and • IDP extension type 505. The first IMSI type Subscription-Id is copied to IMSI. **Note:** There must be an E164 or SIP type Subscription -Id present. Otherwise the message will be rejected.
Service-Identifier	Used as part of the key to look up the service.
Termination-Cause	Use **cmnDebug()** to trace this if this transaction is being traced. Otherwise, ignore.
Requested-Service-Unit	The type of the service unit (derived from which sub-AVP is contained within this one) is placed in IDP extension type 502. The value of the sub-AVP is placed in IDP extension type 501. Multiple unit types are supported. You can perform Basic and MSCC services, with the following provisos: • The units in Initial request are the units for the whole session, that is, you cannot add another unit mid-session. • If one unit fails to be granted, the entire service is denied. • Multiple units are not suitable for use in event based credit control, because the call or dialog with ACS is a one-shot for each type (likely through a Named Event node). If multiple calls are opened to ACS and one of them happens to fail, it is too late for DCA to go back and revoke the successful cases.
Requested-Action	Used as part of the key to look up the service. Also used to determine the next state in the state machine.
Used-Service-Unit	The cumulative total of all the Used-Service-Unit AVPs is copied to ApplyChargingReport.timeNoTariffSwitch (multiplied by 10 to be in deciseconds if the unit type is Time). The variable **dca::StateMachine::totalUsedUnits** is used for storing this information.
Multiple-Services-Indicator	If this is set to MULTIPLE_SERVICES_SUPPORTED then DCA will accept the incoming message and subsequent Multiple-Services-Credit-Control AVPs received in CCR/CCA update and final request messages. **Note:** This parameter will not be mapped to the InitialDP.
Multiple-Services-Credit-Control	This is a grouped AVP that can contain these AVPs: • Requested-Service-Unit • Used Service-Unit • Service-Identifier • Rating Group Requires that Multiple-Services-Indicator AVP has been received with value set to MULTIPLE_SERVICES_SUPPORTED. For multiple services credit control, a single session typically comprises multiple services. Each service is identified by either the Service-identifier

AVP	Action
	or Rating-Group (where no Service-identifier). Requests received are handled as follows: • INITIAL-REQUEST – One IDP is sent for each service. The Diameter session will have multiple INAP dialogs with slee_acs. • UPDATE-REQUEST for a new Service-identifier/Rating-Group – Starts a new service within the session and causes DCA to send another IDP. • UPDATE-REQUEST with no requested-service-unit AVP – Ends a service within the session. • TERMINATION-REQUEST – Ends the whole session. **Note:** • If more than one unit type is received within this AVP, DCA will recognize the used-service-unit AVP in update and termination request messages and will extract the relevant unit used. Typically this will be the unit previously specified in the granted service unit. • If no relevant unit is found then DCA returns CCA (Multiple-Services-Credit-Control(result-Code = DIAMETER_INVALID_AVP_VALUE)).
`Service-Parameter-Info`	Ignored unless mapped to an IDP extension by the AVP mappings in **eserv.config**.
`CC-Correlation-Id`	Ignored unless mapped to an IDP extension by the AVP mappings in **eserv.config**.
`User-Equipment-Info`	Ignored unless mapped to an IDP extension by the AVP mappings in **eserv.config**.
`Proxy-Info`	The stack code in the DIAMETER module handles this.
`Route-Record`	Ignored at present.

CCA

This table describes the mappings for Credit-Control-Answer AVPs.

AVP	Set from
`Session-Id`	The Session-Id AVP of the first message in this transaction. (stored in **dca::StateMachine:: sessionId**
`Result-Code`	Set to DIAMETER_SUCCESS unless otherwise stated.
`Origin-Host`	The stack code in the DIAMETER module sets this.
`Origin-Realm`	The stack code in the DIAMETER module sets this.
`Auth-Application-Id`	Set to 4
`CC-Request-Type`	Leave as the stack default, that is, the value of CC-Request-Type from the corresponding request.
`CC-Request-Number`	Leave as the stack default, that is, the value of CC-Request-Number from the corresponding request.
`User-Name`	Not set
`CC-Session-Failover`	Not set (will default to FAILOVER-NOT-SUPPORTED according to *RFC 4006*)
`CC-Sub-Session-Id`	Set to the value from the corresponding request message, of present.

AVP	Set from
Acct-Multi-Session-Id	Set to the value from the corresponding request message, of present.
Origin-State-Id	Set to **dca::ControlAgent::originStateId**.
Event-Timestamp	Set to the value of the Event-Timestamp AVP from the corresponding request.
Granted-Service-Unit	For session based services, this is ApplyCharging.maxDuration (divided by 10 if the unit type is Time). The unit type is obtained from the "DCA Unit Type" profile tag, if it is available in the ApplyCharging extension profile block. For Requested-Action type DIRECT_DEBIT, in the success case, this is the same as the Requested-Service-Unit AVP in the corresponding request. Otherwise, not present.
Multiple-Services-Credit-Control	DCA will populate the MSCC AVPs in CCA messages with the following sub-AVPs where applicable: • Granted-Units • Rating-Group or Service-Identifier • Result-Code • Time-Quota-Threshold (AVP code 868) • Volume-Quota-Threshold (AVP code 869) • Validity-Time (if applicable) • Final-Unit-Indication (if applicable) **Note:** Both Time-Quota-Threshold AVP and Volume-Quota-Threshold AVP will be sent with: • Vendor_ID 10415 • Quota-Threshold value 0
Cost-Information	For Request-Action type PRICE_ENQUIRY, success case, this comes from the value of extension 603 in the INAP Connect. Otherwise, not set.
Final-Unit-Indication	Final-Unit-Action is set to REDIRECT or TERMINATE depending on the INAP operations received. Redirect-Server is set to the number matched in the redirectNumbers config list or TEL:<Connect destinationRoutingAddress>@<Configured SIP host>.
Check-Balance-Result	This is derived from the type of INAP operation received as described in the *Check balance, with a result of enough credit* (on page 21) scenario.
Credit-Control-Failure-Handling	Set to TERMINATE.
Direct-Debiting-Failure-Handling	Not set. (According to *RFC 4006*, it will default to TERMINATE_OR_BUFFER).
Validity-Time	Set to the configured validity-time for the service in the graceful termination scenarios only. See the *Funds expiry, redirect, top-up and reconnect* (on page 23) scenario.
Redirect-Host	Not set.
Redirect-Host-Usage	Not set.
Redirect-Max-Cache-Time	Not set.
Proxy-Info	The stack code in the DIAMETER module sets this.
Route-Record	Not set at the moment.

AVP	Set from
	If we set this in the future, the stack code in the DIAMETER module will set this.
`Failed-AVP`	Set in some cases when Result-Code != success.

Business Scenarios

Introduction

This topic explains how the flow through the software achieves Diameter server services and also gives more details on the mapping between INAP operations/parameters and Diameter messages/AVPs.

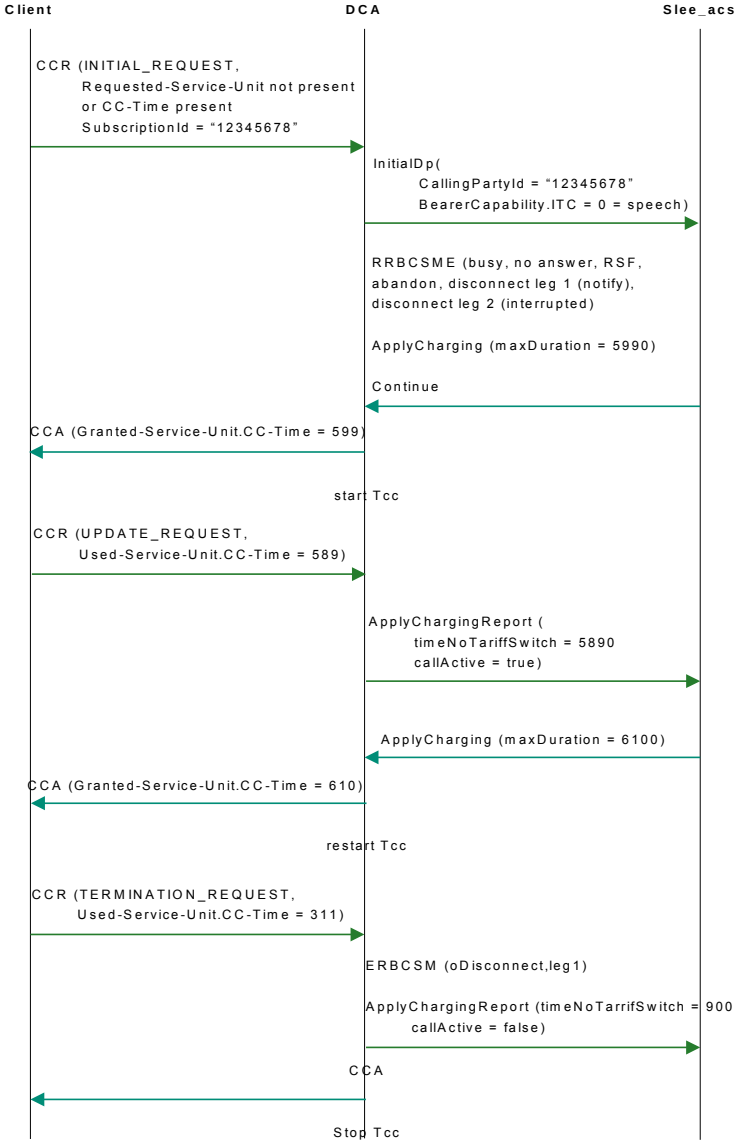
The following scenarios are based on (and named after) the relevant appendixes in *RFC 4006*.

For each business scenario, a message sequence chart is given.

For sample message flows, refer also to the *DCA Messages Flows* chapter in *Sample Message Flows Reference Guide*.

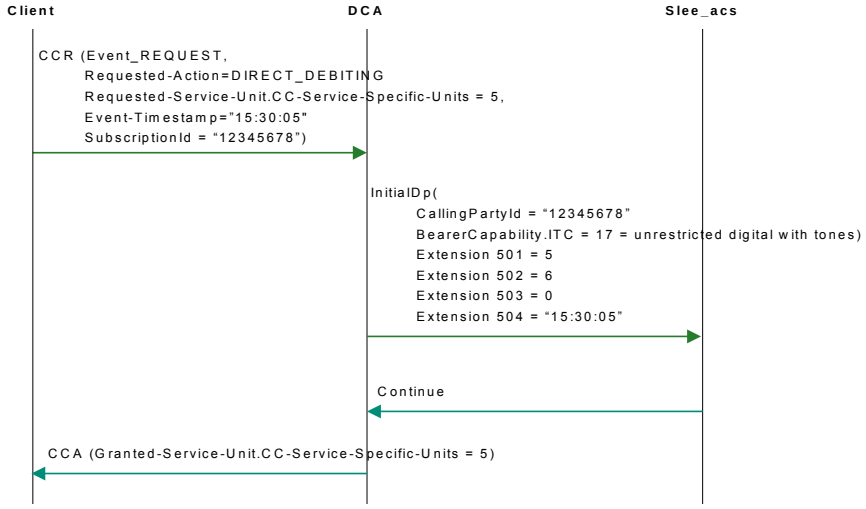
Successful session-based charging, client terminates session

Here is an example successful session-based charging, client terminates session.



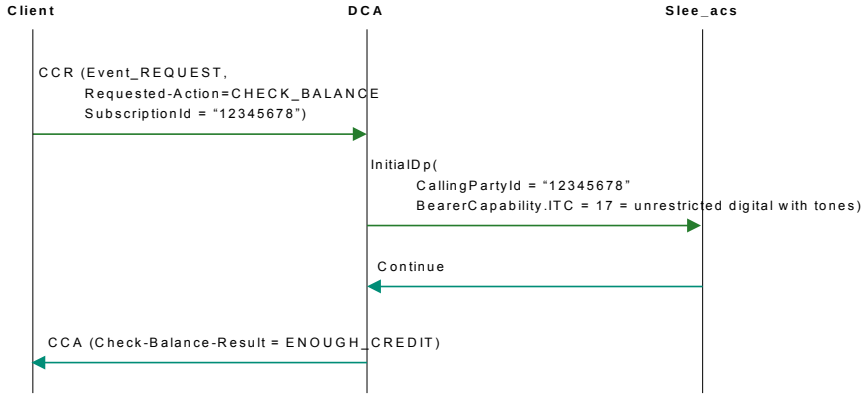
Multimedia messaging direct debit scenario

Here is an example multimedia messaging direct debit scenario.



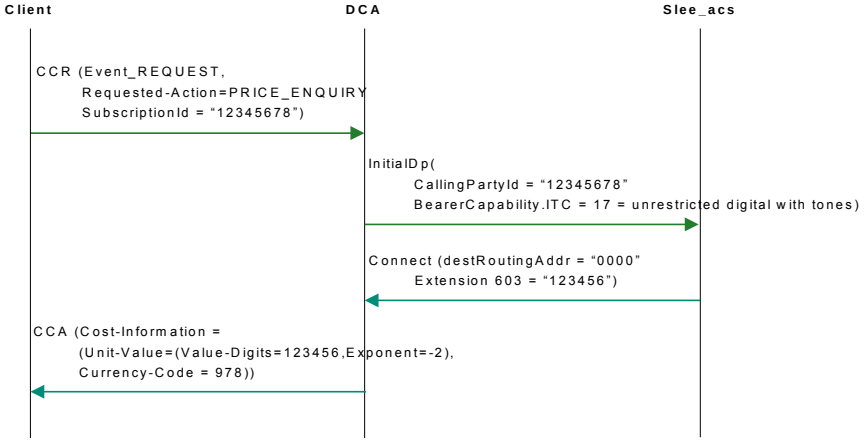
Check balance, with a result of enough credit

Here is an example check balance, with a result of enough credit.



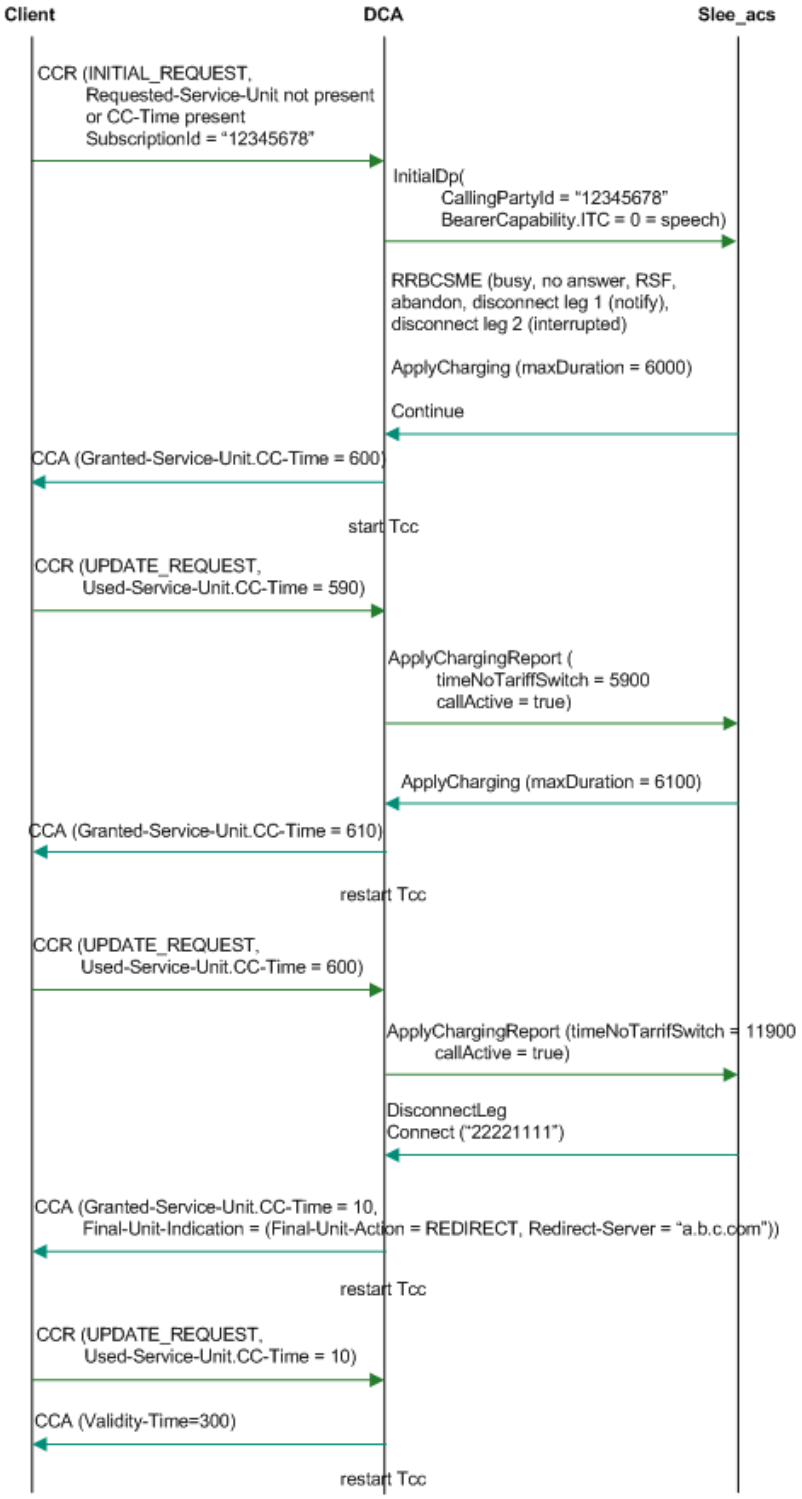
Price enquiry

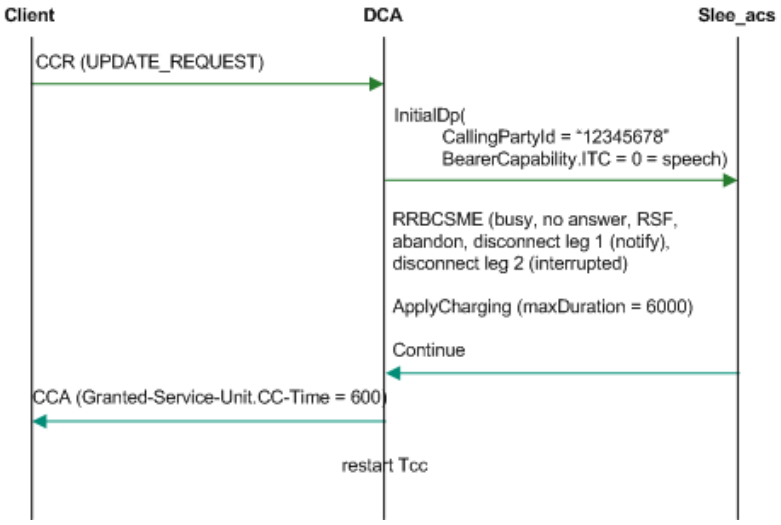
Here is an example price enquiry.



Funds expiry, redirect, top-up and reconnect

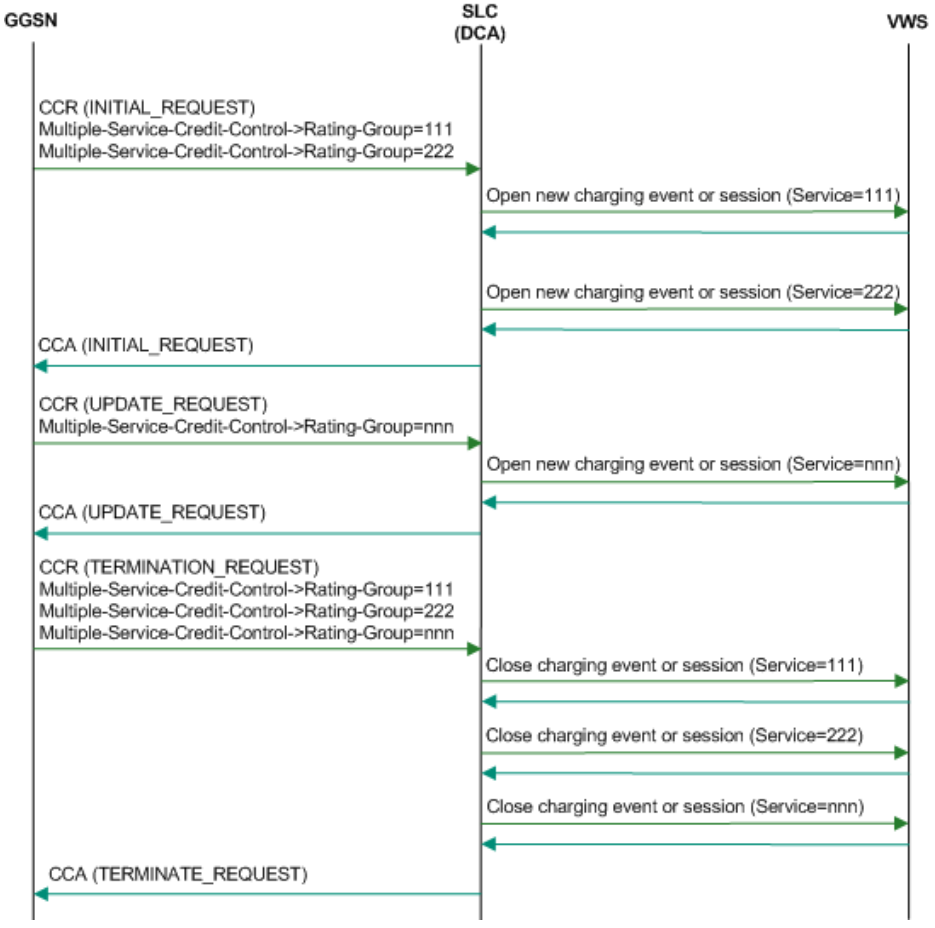
Here is an example funds expiry, redirect, top-up and reconnect.





Multiple services credit control scenario

Here is an example multiple services credit control scenario.



Chapter 3

Background Processes

Overview

Introduction

This chapter explains the process which runs automatically as part of the Oracle Communications Network Charging and Control (NCC) application. This process is started automatically by the SLEE.

In this chapter

This chapter contains the following topics.

dcaResPlugin

This plugin runs on the VWS snd triggers DAP events to trigger HTTP RAR requests to the SLC

Purpose

Sends DAP2 RAR requests to xmlSleeDcaInterface.

dcaResPlugin supplies the callback implementation for reservationAdded. Upon triggering, dcaResPlugin checks the supplied reservation context EDR fields for any DIAMETER call details. If the call is determined to be a DIAMETER call, dcaResPlugin stores in a map (keyed by walletId) the DIAMETER reservations against that wallet (given by clientId and clientMsgId) and the DCA details (DCA_INSTANCE, DCA_SCP, DCA_SESSION, DCA_ORIGIN_HOST), in the following format:

WALT -> [CLID,CMID] [DCA_SCP] [DCA_INSTANCE] [DCA_ORIGIN_HOST] [DCA_SESSION]

For example:

1 -> [1, 1] [slc1.example.com] [dcaIf1] [MIPT-TEST] [MIPT-TEST;1500000000;1]

2 -> [1, 2] [slc1.example.com] [dcaIf1] [MIPT-TEST] [MIPT-TEST;1500000001;1]

3 -> [2, 3] [slc2.example.com] [dcaIf2] [MIPT-TEST] [MIPT-TEST;1500000004;1]

During the call, an out-of-band balance update event (for example, account top-up) may be triggered by the subscriber. The plugin (in addition to the new reservation callbacks) supplies methods for existing wallet/balance/bucket callbacks.

For example, if the balance update callback is triggered on a wallet previously determined as subject to an open DIAMETER reservation, the plugin logic creates a DAP2 notification event to be populated with relevant DCA session data (DCA_INSTANCE, DCA_SCP, DCA_SESSION, DCA_ORIGIN_HOST) obtainable via reservation map lookup.

Startup

This process is started automatically by the SLEE. For more information see *SLEE.cfg Configuration* (on page 7).

Default Configuration

```
CCS = {
    # dcaResPlugin.so config
    dcaResPlugin = {
       # Operation set for RAR notifications
       dapOperationSet = "RAR"
    }
}

BE = {
  plugins = [
     # other plugins
     "dcaResPlugin.so"
  ]
}
```

Alarms

The following alarms can be raised by dcaVWARSPlugin:

- ERROR [960601] Failed to read plugin config: <string>
- NOTICE [960602] Loaded plugin config.
- ERROR [960601] Failed to read plugin config: <string>
- ERROR [960603] Cannot read Tariff Handler data.
- ERROR [960604] Cannot read cascadeBalances in Tariff Handler data.
- ERROR [960605] Cannot read cascadeXBalances in Tariff Handler data.
- NOTICE [960606] Cannot read CDR tags in Tariff Handler data.
- WARNING [960607] Incomplete RAR tags in Tariff Handler CDR data.

diameterControlAgent Process

Purpose

The diameterControlAgent executable is a SLEE interface which converts between Diameter messages and CAP3 operations to enable a Diameter client to communicate with a CAP3 SCF.

Startup

This process is started automatically by the SLEE. For more information see *SLEE.cfg Configuration* (on page 7).

DIAMETER Configuration Structure

Here is the high-level structure of the `DIAMETER` configuration section of the **eserv.config** file.

```
DIAMETER = {
    DCADefaults = {
        DCADefaults_parameters
    }

    DCAInstances = [
    # First Instance
    {
    NumberRules = [
        NumberRules_parameters
    ]
        DCAIinstances_parameters
    ]

    RedirectNumberMappings = [
    {
        RedirectNumberMappings_parameters
    }
    ]

    Tracing = {
        Tracing_parameters
    }

        Services = [
        {
            Services_parameters
        }
        ]

        DiameterServer = {
            DiameterServer_parameters
        }

    } # end of First Instance

    ] # end of DCAInstances section


    PeerSchemes = [
    # First Scheme
    {
        schemeName = "SchemeA"

        Peers = [

        {
            peerhost1_parameters
        }
        {
            peerhost2_parameters
        }

        }

    {
        schemeName = "SchemeB"
            SchemeB_parameters
        }

    ] # End of PeerSchemes section
```

Failure

If the diameterControlAgent fails, no Diameter messages will be processed.

xmlSleeDcaInterface

Purpose

Converts DAP2 RAR requests to SLEE RAR Events and forwards to diameterControlAgent.

Accepts dcaRarReq messages and uses the data contained therein to construct a diameterSleeEvent message.

Startup

This process is started automatically by the SLEE. For more information see *SLEE.cfg Configuration* (on page 7).

Default Configuration

```
DIAMETER = {
   #xmlSleeDcaIF
   DCA = {
      # xmlSleeDcaIF listens for HTTP connections
      listenPort = 3088

      # Incoming connection detection polling timer
      # (microseconds)
      pollTime = 100000

      # Automatic periodic config reload time?
      # (seconds)
      # 0 = disabled
      reloadInterval = 0
   }
}
```

Alarms

The following alarms can be raised by xmlSleeDcaInterface:

- WARNING [960501] Terminated with INTERFACE_END.
- WARNING [960502] Terminated with INTERFACE_KILL.
- NOTICE [960503] Reread config management event.
- ERROR [960504] accept() failed. duplicate fd: %d
- CRITICAL [960505] Failed to get management event type from SLEE.
- NOTICE [960506] Received unknown EventType from SLEE: <String>.
- CRITICAL [960507] Failed to get SLEE API handle: <String>.
- CRITICAL [960508] Failed to initialise Xerces parser..
- ERROR [960509] sigaction() failed: <String>.
- CRITICAL [960510] Failed to read config: <String>.
- NOTICE [960511] xmlSleeDcaIF is now running.
- ERROR [960512] Failed to reread config: <String>.
- WARNING [960521] Outstanding request on fd: <Int>.
- NOTICE [960522] Connection closed by foreign host (fd: <int>).

- WARNING [960523] read() error (fd: <Int>) : <String>.
- WARNING [960524] Received invalid HTTP request.
- ERROR [960525] XMLException: <String>.
- ERROR [960526] SAXParseException: <String>.
- ERROR [960527] UnknownException: <String>.
- ERROR [960528] 503 Service Unavailable: Unable to send RAR event.
- ERROR [960529] 503 Service Unavailable: Could not create RAR event.
- NOTICE [960531] No existing entry found in cache for this interface.
- WARNING [960532] Unable to get SLEE handle for DCA interface.
- ERROR [960533] SLEE error: Could not sendEvent() to DCA.
- ERROR [960534] Unable to create a DiameterSleeEvent.

DCADefaults Configuration Section

Example DCADefaults Configuration in eserv.config File

Here is an example `DCADefaults` section of the `DIAMETER` configuration in the **eserv.config** file.

```
DCADefaults = {
    sleeServiceKey = 1234
    inapServiceKey = 1234
    maxSessionLengthAfterFinalUnitIndicationsSeconds = 14340
    tcc = 3600
    gracefulTerminationValidityTime = 300
    validityTime = 30
    systemErrorResultCode = 5012
    invalidMessageSequenceResultCode = 5012
    itc = "udi"

    AvpMappings = [
    {
        AvpCodes = [
        {
            avpCode = 1234
            mandatory = true
            vendorId = "16747"
        }
        ]

        avpFormat = "OctetString"
        sipScheme = "sip"
        extensionType = 1234
        extensionFormat = "inapnumber"

        conversion = [
            { internal = 1, external = 5030 }
            { internal = 16, external = 2001 }
            { internal = 17, external = 3004 }
            { internal = 42, external = 5006 }
            { internal = 111, external = 3001 }
        ]
        mappingTypes = ["InitialRequest", "InitialResponse", "EventRequest",
        "EventResponse"]
    }
    ]
}
```

DCADefaults Parameters

The following parameters are used as defaults if not specified in a Service. They are found within the `DCADefaults = { }` statement.

avpMappings

Syntax: `avpMappings = [mappings_parameters]`

Description: The default service AVP mappings.

Optionality: Mandatory

Notes: See *AvpMappings Parameters* (on page 32).

gracefulTerminationValidityTime

Syntax: `gracefulTerminationValidityTime = seconds`

Description: The number of seconds granted for the user to top up the account during graceful termination. Refer to *RFC 4006* A.7.

Type: Integer

Optionality: Optional

Allowed: in seconds

Default:

Notes: Not present means no graceful termination.

Example: `gracefulTerminationValidityTime = 300`

inapServiceKey

Syntax: `inapServiceKey = value`

Description: The INAP Key value

Type: Integer

Optionality: Mandatory

Allowed: Any 32 bit integer

Example: `inapServiceKey = 1234`

invalidMessageSequenceResultCode

Syntax: `invalidMessageSequenceResultCode = code`

Description: The error code for an invalid message sequence result, for example, if TERMINATION_REQUEST is the first message.

Type: Integer

Optionality: Mandatory

Allowed:

Default: 5012 [Diameter unable to comply]

Notes: See Part 7.1 of *RFC 3588* and Part 9 of *RFC 4006* for a list valid codes.

Example: `invalidMessageSequenceResultCode = 5012`

itc

Syntax: `itc = infoTransferCapability`

Description: The Bearer Capability Information Element (Q.931 section 4.5.5) contains an Information Transfer Capability (ITC) field that is set automatically by DCA when DCA triggers ACS.

This parameter overrides the ITC value within the Bearer Capability Information Element.

For more details, please see `itc` (on page 66) parameter under the Services section.

Type: Integer, or string

Optionality: Optional

Allowed:

Default:

Notes: If automatic setting of ITC is required, then this parameter should be absent.

Example:
```
itc = 16
```
or
```
itc = "3.1kHzAudio"
```
or
```
itc = 0x10
```

mappingTypes

Syntax: `mappingTypes = ["`*`mapping_types`*`"]`

Description: Specifies the cases that the mapping applies to.

Type: String Array

Optionality: Optional

Allowed:

Default:

Notes: For more details, please see the `mappingTypes` (on page 74) parameter under the Services section.

Example:
```
mappingTypes = ["InitialRequest", "UpdateRequest",
"EventRequest", "TerminateRequest"]
```

maxSessionLengthAfterFinalUnitIndicationsSeconds

Syntax:
```
maxSessionLengthAfterFinalUnitIndicationsSecon
ds = secs
```

Description: The maximum number of seconds that a session can last after the Final-Unit-Indication AVP has been sent to the client.

Type: Integer

Optionality: Optional (default used if not set).

Allowed: A valid integer.

Default: 14340

Example:
```
maxSessionLengthAfterFinalUnitIndicationsSecon
ds = 14340
```

sleeServiceKey

Syntax `sleeServiceKey = value`

Description: The Service Key value

Type: Integer

Optionality: Mandatory

Allowed: *Refer to SLEE Technical Guide*

Default: N/A

Example: `sleeServiceKey = 1234`

systemErrorResultCode

Syntax: `systemErrorResultCode = code`
Description: The error code for a system error
Type: Integer
Optionality: Mandatory
Allowed:
Default: 5012 [Diameter unable to comply]
Notes: See Part 7.1 of *RFC 3588* and Part 9 of *RFC 4006* for a list valid codes
Example: `systemErrorResultCode = 5012`

tcc

Syntax: `tcc = value`
Description: The Session supervision timer timeout
Type: Integer
Optionality: Mandatory
Allowed: number of seconds
Default: 3600
Notes: Refer to *RFC 4006*.
Example: `tcc = 3600`

validityTime

Syntax: `validityTime = seconds`
Description: The validity time in seconds of granted units. Results in Validity-Time AVP being placed in CCA.
Type: Integer
Optionality: Optional (default used if not set).
Allowed:
Default: -1 (Not included)
Notes:
Example: `validityTime = 30`

AvpMappings Parameters

The following parameters are used for AVP mappings. They are all found within an `AvpMappings = [ ]` array.

You can set up as many AVP mappings as required.

Within this section you can specify AVP codes for mapping. They are all found within an `AvpCodes = [ ]` array.

There MUST be one specified for the base AVP, plus list all extras for grouped AVPs.

AVP Format to Extension Type

This table shows the allowable conversion of AVP format to the Extension type.

AVP Format	Extension Type	Notes
OctetString	INAP Number	Not allowed
Time	INAP Number	Not allowed
String/OctetString	Integer	• Must be ASCII digits • Converts to Integer
String	INAP Number	Must be Hex digits
Integer	String/Octets	Converts to string
Time	Integer	Number of seconds since 1 January 1970
Time	String/Octets	In format *YYYYMMDDHH24mmss*

AVP Casting

Casting between the AVP format and the encoded extension format is supported only for encoded extension formats with variable sizes. All other encoded extension formats are fixed in size and cannot be casted. In the **eserv.config** file, encoded extension formats are defined by the `profileFormat` parameter. See *profileFormat* (on page 73) for more information.

This table shows the `profileFormat` values that have variable sizes (which can be casted) versus fixed sizes (which cannot be casted).

Size Type	profileFormat Value
Variable size	LNSTRING, STRING, NSTRING, and RAW_DATA
Fixed size	INTEGER, INTEGER64, UINTEGER, UINTEGER64, TIME, BOOLEAN, and ARRAY

For example, the following configuration casts an OctetString AVP format (1 byte) as a STRING encoded extension format (1 byte), which is supported:

```
{
   AvpCodes = [
      {
         avpCode = 21 # RAT-Type
         mandatory = true
         vendorId = 10415
      }
   ]
   avpFormat = "OctetString"
   encodedExtension = {
   profileTag = 6760105
   profileFormat = "STRING"
}
```

In contrast, the following configuration attempts to cast an OctetString AVP format (1 byte) as an INTEGER encoded extension format (4 bytes), which is not supported:

```
{
   AvpCodes = [
      {
         avpCode = 21 # RAT-Type
         mandatory = true
         vendorId = 10415
       }
   ]
   avpFormat = "OctetString"
```

```
    encodedExtension = {
    profileTag = 6760105
    profileFormat = "INTEGER"
}
```

avpCode

Syntax: `avpCode = code`

Description: The AVP code for this AVP.

Type: Integer

Optionality: Mandatory

Notes: This parameter is an element of the `AvpCodes` parameter array.

Example: `avpCode = 1234`

AvpCodes

Syntax: AvpCodes = [avpcodes]

Description: The AVP code[s] specifying the AVP.

Type: Array

Optionality: Optional

Allowed:

Default:

Notes: FULLY-QUALIFIED AVP CODE:

If more than 1 element is specified in this array, the AvpCodes refer to a Fully-Qualified "path" into the AVP hierarchy.

RELATIVE AVP CODE:

If only a single entry is specified and if this "AvpCodes" is used within the "AVPs" Array Section, each element in "AvpCodes" is relative and all the nested AVPs collectively form the complete "path" to specifying the AVP within the hierarchy.

Example:

```
AvpCodes = [
 {
 avpCode = 1234
 mandatory = true
 vendorId = "16747"
 }
]
```

avpFormat

Syntax: `avpFormat = "format"`

Description: The format of the AVP.

Type: String

Optionality: Mandatory

Allowed: Allowed values are:

- "OctetString"
- "Integer32"
- "Integer64"
- "Unsigned32"
- "Unsigned64"
- "Address"

- "Time"
- "UTF8String"
- "DiameterIdentity"
- "DiameterURI"
- "Enumerated"
- "Grouped" – Only valid if used in *Service Specific AVP Mappings* (on page 85)

Example: `avpFormat = "OctetString"`

conversion

Syntax: `conversion = [mappings]`

Description: For Integer type formats, you can use this parameter to define a conversion table (for outbound mapping) for further mapping of internal (typically INAP cause, or acsProfile values) to external (that is, Diameter AVP) values.

Type: Parameter array

Optionality: Optional

Allowed:

Default:

Notes:

Example: This example is mapping from internal INAP Cause codes to its Diameter Result-Code.

```
conversion = [
 { internal = 1, external = 5030 }
 { internal = 16, external = 2001 }
 { internal = 17, external = 3004 }
 { internal = 42, external = 5006 }
 { internal = 111, external = 3001 }
]
```

extensionFormat

Syntax: `extensionFormat = "format"`

Description: The format of the extension in ACS.

Type: String

Optionality: Optional

Allowed:

- "inapnumber"
- "asn1integer"
- "octets"
- "encoded" – Only valid if used in Service Specific AVP Mappings. See *encodedExtension* (on page 71).

Notes:

Example: `extensionFormat = "inapnumber"`

extensionType

Syntax: `extensionType = type`

Description: The InitialDP extension type

Type: Integer

Optionality: Mandatory

Notes: Cannot be a pre-defined INAP extension
nonProfile Encoded Extensions should be considered deprecated

Example: `extensionType = 1234`

external

Syntax: `external = value`

Description: The external value to be put into the AVP to be sent.

Type: Integer

Optionality: Optional

Allowed:

Default:

Notes: Member of the `conversion` section.

Example: `external = 5030`

internal

Syntax: `internal = value`

Description: The internal value (typically INAP cause or acsProfile values) from ACS.

Type: Integer

Optionality: Optional

Allowed:

Default:

Notes: Member of the `conversion` section.

Example: `internal = 1`

mandatory

Syntax `mandatory = true|false`

Description: Whether the AVP code is mandatory

Type: Boolean

Optionality: Optional

Allowed: true, false

Default: false

Notes: This parameter is an element of the `AvpCodes` parameter array.

Example : `mandatory = true`

noa

Syntax: `noa = value`

Description: The Nature of Address (NOA) for the INAP number. If a SIP Address AVP telephone number is not international, the NOA of a mapped INAP Number will be set to the value specified in this parameter.

Type: Integer

Optionality: Optional (default used if not set).

Allowed:

- 1 – Subscriber
- 2 – Unknown
- 3 – National
- 4 – International

Default: 4

Notes: The NoA field of an INAP Number will be set to International (4) if the mapped SIP address AVP telephone number begins with '+'.

Example: `noa = 2`

sipScheme

Syntax: `sipScheme = "sip_scheme"`

Description: Indicates that the AVP contains a SIP address conforming to the URL scheme configured.

Type: String

Optionality: Optional (default used if not present).

Allowed: Allowed values:

- "sip"
- "tel"

Default: *parameter not present*. No mapping done.

Notes: If specified, the SIP address (if found in the AVP) will be extracted and used in the mapped field sent to ACS. See *sipScheme example configurations* (on page 37).

The following parameter must be set:

- `avpFormat = "UTF8String"` - if not set, AVP mapping will be ignored.

If a sipScheme is used, the destination is restricted. It can only be mapped to one of the following:

- `inapField`
- `extensionFormat` (of "`inapnumber`", or "`octets`")

If not specified, this indicates that the AVP does not contain a SIP address, so no address information will be extracted from the AVP.

Example: `sipScheme = "sip"`

sipScheme example configurations

The following examples illustrate the use of the `sipScheme` parameter in the `AvpMappings` configuration, and the resulting mappings.

Example 1

Map AVP 1000 to the calledPartyNumber field in IDP. For example, with "sip:+12125551212@phone2net.com;tag=887s", the digits**+12125551212** are mapped to calledPartyNumber.

```
{
    AvpCodes = [
        {
            avpCode = 1001
        }
    ]
    avpFormat = "UTF8String"
    sipScheme = "sip"
    inapField = ["calledPartyNumber"]
    mappingTypes = ["InitialRequest"]
}
```

Example 2

Map AVP 2000 to extension 7890 of type InapNumber. For example, with "tel:+358-555-1234567", the digits**+358-555-1234567** are mapped to extension 7890.

```
{
    AvpCodes = [
        {
            avpCode = 2000
        }
    ]
    avpFormat = "UTF8String"
    sipScheme = "tel"
    extensionFormat = "inapnumber"
    extensionType = 7890
    mappingTypes = ["InitialRequest"]
}
```

Example 3

Map AVP 2000 to extension 8000 of type InapNumber. For example, with "tel:555-1234567", the digits **555-1234567** are mapped to extension 8000. Because the number is not internationalized (no leading '+'), you must set the Nature of Address (`noa` parameter) to the configured value of 2.

```
{
    AvpCodes = [
        {
            avpCode = 2000
        }
    ]
    avpFormat = "UTF8String"
    sipScheme = "tel"
    noa = 2
    extensionFormat = "inapnumber"
    extensionType = 8000
    mappingTypes = ["InitialRequest"]
}
```

vendorId

Syntax: `vendorId = "id"`

Description: The vendor specific AVP, if present.

Type: String

Optionality: Optional

Notes: This parameter is an element of the `AvpCodes` parameter array.

Example: `vendorId = "16747"`

DCAInstances Configuration Section

Introduction

Due to the size and complexity of the `DCAInstances` configuration, the description is broken down into the following topics:

DCAInstances configuration structure

Here is the high level structure of the configuration of an instance in the `DCAInstances` section of the `DIAMETER` configuration in the **eserv.config** file.

```
DCAInstances = [

# First Instance
{
NumberRules = [
    NumberRules_parameters
]

dummyDestination = "0000"
systemCurrencyCode = 978
systemCurrencyExponent = -2

multipleServicesRatingGroup = 0

allowDefaultRatingGroup = false

customDefaultUnits = false

dontDiscardRatingGroupInResponse = true

SIPDomain = "SIP_Domain"
SIPPrefix = "SIP_prefix"

ignoreRSU = true
}
rarHandlingEnabled = false
rarClientTimeout = 30
rarMaxRetry = None



RedirectNumberMappings = [
{
    RedirectNumberMappings_parameters
}
]

Tracing = {
    Tracing_parameters.
}

instanceName = "dcaIf"
scheme = "SchemeA"
systemErrorResultCode = 5012
invalidMessageSequenceResultCode = 5012
sessionBasedDuplicateDetection = true
returnServiceResultCodeInRoot = false
ggsnSupportsFinalUnitIndication = true
ccDuplicateStoreSize = 20
maxAnswerReorder = 2
roundingThreshold = "0.5"
roundingDetail = "ceil"
SubscriptionIdTypes = [
    0,
    2,
    1
]
```

```
Services = [
{
    Services_parameters.
}
]

DiameterServer = {
    DiameterServer_parameters
}

} # end of First Instance

] # end of Instances section
```

Note: Default settings are specified at installation time.

DCAInstances Parameters

Here are the parameters for the `DCAInstances` section.

`allowDefaultRatingGroup`

Syntax: `allowDefaultRatingGroup = true|false`

Description: Whether or not to use the default Rating Group.

Type: Boolean

Optionality: Optional (default used if not set).

Allowed: If set to:

- true — and if no Service-Identifier AVP or Rating-Group AVP is received in the initial request CCR, DCA will use the Default Rating Group defined in `multipleServicesRatingGroup`.
- false — DCA will not use the default Rating Group but instead will wait for the Service-Identifier AVP or Rating-Group AVP in the subsequent request.

Default: false

Notes:

Example: `allowDefaultRatingGroup = false`

`ccDuplicateStoreSize`

Syntax: `ccDuplicateStoreSize = value`

Description: The number of credit-control messages to maintain, when checking for duplicates.

Type: Integer

Optionality: Optional (default used if not set).

Allowed:

Default: 20

Notes: Requires `sessionBasedDuplicateDetection` to be set to true.

Example: `ccDuplicateStoreSize = 10`

customDefaultUnits

Syntax: `customDefaultUnits = true|false`

Description: Applies only when multiple service credit control (MSCC) is in use. Try to accommodate operator-specific ways of handling default units in the following circumstances:

An initial request with an MSCC having:

- A Used-Service-Unit AVP containing CC-Time = 0
- No Requested-Service-Unit AVP

An update request with an MSCC having:

- An Used-Service-Unit AVP containing CC-Time = <actual used>
- No Requested-Service-Unit AVP

Type: Boolean

Optionality: Optional (default used if not set)

Allowed:

- true – DCA behaves as if the RSU had been specified with the unit type of CC-Time.
- false – DCA behaves as it normally does if no RSU has been specified.

Default: false

Notes:

Example: `customDefaultUnits = true`

dummyDestination

Syntax: `dummyDestination = "num"`

Description: Used as the Called Party Number in the InitialDP.

Type: Number string

Optionality: Mandatory

Allowed: Digits

Default: 0000

Notes: DCA requires a destination; however we do not use it.

Example: `dummyDestination = "0000"`

dontDiscardRatingGroupInResponse

Syntax: `dontDiscardRatingGroupInResponse = true|false`

Description: If set to true, and a Service-Identifier AVP, and a Rating-Group AVP are both received in the request AVP (for MSCC), then the Rating-Group AVP is returned in the CCA response. Otherwise the Rating-Group AVP is not returned.

Type: Boolean

Optionality: Optional (default used if not set).

Default: false

Example: `dontDiscardRatingGroupInResponse = false`

ggsnSupportsFinalUnitIndication

Syntax: `ggsnSupportsFinalUnitIndication = true|false`

Description: Whether or not the GGSN supports final unit indication AVPs.

Type: Boolean
Optionality: Optional (default used if not set).
Allowed: true, false
Default: true
Notes: A false value implies a non-compliant GGSN.
Example: `ggsnSupportsFinalUnitIndication = true`

ignoreRSU

Syntax: `ignoreRSU = true|false`
Description: Specifies whether DCA ignores any unit types in Requested-Service-Unit AVPs that are included in a request. This parameter applies to session request types only.
Type: Boolean
Optionality: Optional (default used if not set)
Allowed:
- true – DCA ignores any unit types in Requested-Service-Unit AVPs that are included in a request by treating them like empty RSUs.
- false – DCA recognizes all Requested-Service-Unit AVPs in requests.

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

instanceName

Syntax: `instanceName = "name"`
Description: The unique identifying name for this instance.
Type: String
Optionality: Mandatory
Allowed: This must match the interface name in SLEE.cfg. See *SLEE.cfg Configuration* (on page 7).
Default: no default
Notes:
Example: `instanceName = "dcaIf"`

invalidMessageSequenceResultCode

Syntax: `invalidMessageSequenceResultCode = code`
Description: The error code for an invalid message sequence result, for example, if TERMINATION_REQUEST is the first message.
Type: Integer
Optionality: Mandatory
Allowed:
Default: 5012 [Diameter unable to comply]
Notes: See Part 7.1 of *RFC 3588* and Part 9 of *RFC 4006* for a list valid codes.
Example: `invalidMessageSequenceResultCode = 5012`

maxAnswerReorder

Syntax: `maxAnswerReorder = number`
Description: The maximum number of answers to consider for reordering, before giving up, and answering.

Type: Integer
Optionality: Optional (default used if not set).
Allowed:
Default: 10
Notes:
Example: `maxAnswerReorder = 2`

multipleServicesRatingGroup

Syntax: `multipleServicesRatingGroup = value`
Description: Rating Group value used to populate Rating Group AVPs in Multiple Services Credit Control AVPs.
Type: Integer
Optionality: Optional (default used if not set).
Allowed:
Default: 0
Notes:
Example: `multipleServicesRatingGroup = 3`

originHostMustBeFQDN

Syntax: `originHostMustBeFQDN = true`
Description: Sets whether the Origin-Host needs to be a fully qualified domain name.
Type: Boolean
Optionality: Optional (default used if not set)
Allowed: `true` = DCA will reject messages
`false` = DCA will accept messages
Default: `true`
Notes: With this parameter set to true, DCA will reject messages from an Origin-Host which is not a fully qualified domain name. If this parameter is set to false, DCA will accept messages regardless of the Origin-Host parameter.
Example: `originHostMustBeFQDN = true`

rarClientTimeout

Syntax: `rarClientTimeout = int`
Description: Specifies the time, in seconds, that the DCA will wait for an RAA response from the Diameter client
Type: Integer
Allowed: >=0
Default: 30
Notes: Set `rarClientTimeout` to 0 (zero) to disable timeouts.
Example: `rarClientTimeout=30`

rarHandlingEnabled

Syntax: `rarHandlingEnabled = true|false`
Description: Determines whether RAR Handling is enabled in DCD
Type: Boolean

Optionality: Optional (default used if not set)
Allowed: true, false
Default: false
Notes: When `rarHandlingEnabled` is set to **`false`**, the DCD responds to an RAR with an RAA containing the corresponding result code set in the `rarResultCode` parameter
Example: `rarHandlingEnabled = false`

rarMaxRetry

Syntax: `rarMaxRetry = int`
Description: Specifies the maximum number of times that the DCA will attempt to re-transmit an RAR to the Diameter client
Type: Integer
Allowed: >=1
Default: None
Notes: A single re-transmit is allowed per timeout.

returnServiceResultCodeInRoot

Syntax: `returnServiceResultCodeInRoot = true|false`
Description: Whether or not service result codes should always be returned in the root level (against the dictates of RFC 4006).
Type: Boolean
Optionality: Optional (default used if not set).
Allowed: true, false
Default: false
Notes: A true value implies a non-compliant GGSN.
Important: This can only apply (work sensibly) where there is a single service.
Example: `returnServiceResultCodeInRoot = false`

roundingThreshold

Syntax: `roundingThreshold = "fraction"`
Description: The threshold for rounding fractional unit values to integers.
Fractional parts less than this amount are rounded down, fractional parts greater than this amount are rounded up.
Type: String containing a float
Optionality: Optional (default used if not set).
Allowed: Between 0.0 and 1.0
Default: "0.5"
Notes: If you want to truncate, set this to "1".
If you want to round all non-integers upwards, then set this to "0" and use "floor" for `roundingDetail` parameter).
Example: `roundingThreshold = "0.5"`

roundingDetail

Syntax: `roundingDetail = "rounding"`
Description: The direction to round the number when the fractional part equals the `roundingThreshold`.

Type: String
Optionality: Optional (default used if not set)
Allowed: Allowed values are:
- "ceil" means upwards
- "floor" means downwards

Default: "ceil"
Notes:
Example: `roundingDetail = "ceil"`

scheme

Syntax: `scheme = "name"`
Description: The name of which scheme configuration this instance uses.
Type: String
Optionality: Mandatory
Allowed: This must be a `SchemeName` from the `PeerSchemes` section.
Default: no default
Notes:
Example: `scheme = "SchemeA"`

sessionBasedDuplicateDetection

Syntax: `sessionBasedDuplicateDetection = true|false`
Description: Whether to use CC-Request-Number and Session-Id for duplicate detection for session based services, as specified in *RFC 4006*. Otherwise the algorithm from *RFC* 3588 is used.
Type: Boolean
Optionality: Mandatory
Allowed: true, false
Default: true
Notes: Set to false if the clients do not implement this mechanism from *RFC 4006*.
Example: `sessionBasedDuplicateDetection = true`

SIPDomain

Syntax: `SIPDomain= "domain"`
Description: The SIP domain for telephone redirections.
Type: String
Optionality: Mandatory
Allowed:
Default: no default
Notes: Used when no `redirectNumberMapping` exists for the given value.
Example: `SIPDomain = "oracle.com"`

SIPPrefix

Syntax: `SIPPrefix= "prefix"`
Description: The SIP prefix for telephone redirections.

Type: String
Optionality: Mandatory
Allowed:
Default: "tel+"
Notes: Used in redirect-server-address when connect received from SLEE_acs.
Used when no `RedirectNumberMapping` exists for the given value.
Example: `SIPDomain = "tel+"`

SubscriptionIdTypes

Syntax: `SubscriptionIdTypes = [Ids]`
Description: If there is more than one Diameter Subscription-ID in the request, the Subscription-ID with a Subscription-ID-Type nearest the top of this list is used.
Type: Integer array
Optionality: Optional (default used if not set).
Allowed: Subscription-ID-Types defined in *RFC 3588* are:

0 END_USER_E164
The identifier is in international E.164 format (for example, MSISDN), according to the ITU-T E.164 numbering plan defined in [E164] and [CE164].

1 END_USER_IMSI
The identifier is in international IMSI format, according to the ITU-T E.212 numbering plan as defined in [E212] and [CE212].

2 END_USER_SIP_URI
The identifier is in the form of a SIP URI, as defined in [SIP].

3 END_USER_NAI
The identifier is in the form of a Network Access Identifier, as defined in [NAI].

4 END_USER_PRIVATE
The Identifier is a credit-control server private identifier.

Default: 0, 2, 1
Notes: If an entry in the list is not matched, then a Diameter error is returned.
Example:
```
SubscriptionIdTypes = [
    0,
    2,
    1
]
```

systemCurrencyCode

Syntax: `systemCurrencyCode = code`
Description: The ISO 4217 code of the currency.
Type: Integer
Optionality: Mandatory
Allowed: ISO 4217 code of the currency.
Default: 978 (Euro)
Notes:
Example: `systemCurrencyCode = 978`

systemCurrencyExponent

Syntax: `systemCurrencyExponent = code`
Description: The exponent value of small units for a big unit in the currency.
Type: Integer
Optionality: Mandatory
Allowed:
Default: -2 (100 small units for every big unit.)
Notes:
Example: `systemCurrencyExponent = -2`

systemErrorResultCode

Syntax: `systemErrorResultCode = code`
Description: The error code for a system error
Type: Integer
Optionality: Mandatory
Allowed:
Default: 5012 [Diameter unable to comply]
Notes: See Part 7.1 of *RFC 3588* and Part 9 of *RFC 4006* for a list valid codes
Example: `systemErrorResultCode = 5012`

NumberRules Parameters

The following parameters define the number normalization rules for DCA. They are found within `NumberRules = [ ]`.

This section is optional.

Example NumberRules configuration

Here is an example `NumberRules` section of the `DCAInstances` configuration.

```
NumberRules = [
    { prefix="25", fromNoa=3, min=8, max=9, remove=0, prepend="0" }
    { fromNoa=4, remove=0, prepend="00" }
    { prefix="027", min=9, remove=1, resultNoa=3 }
    { prefix="00", min=5, remove=2, prepend="", resultNoa=4 }
]
```

fromNoa

Syntax: `fromNoa = int`
Description: Used when attempting to match the nature of address (NoA) number contained in a message. If there is a match, the `fromNoa` part of the number rule is evaluated.
Type: Integer
Optionality: Required
Allowed:
- 2 – For unknown NoAs
- 3 – For national NoAs
- 4 – For international NoAs

Notes: If you omit `fromNoa` from the `NumberRules` parameter section, then no matching rule will be found.
Example: `fromNoa = 3`

max

Syntax: `max = num`

Description: Specifies the maximum number of digits a number may contain. To meet the max part of the number rule, the number of digits in the number must be equal to or less than the value of `max`.

Type: Integer

Optionality: Optional (default used if not set)

Default: 999

Example:
```
max = 9
```

min

Syntax: `min = num`

Description: Specifies the minimum number of digits a number may contain. To meet the min part of the number rule, the number of digits in the number must be equal to or greater than the value of `min`.

Type: Integer

Optionality: Optional (default used if not set)

Default: 0

Notes: The value of the `min` parameter must be greater than or equal to the value of the `remove` (on page 49) parameter.

Example:
```
min = 5
```

prefix

Syntax: `prefix = "pref"`

Description: Contains a digit or digits. Used to attempt to match the first digit or digits of a prefix number with the specified value. If the digit or digits match, the prefix part of the number rule is met.

Type: String

Optionality: Optional

Allowed: One or more decimal digits

Notes: This parameter is an element of the `NumberRules` parameter array.

Example:
```
prefix = "25"
```

prepend

Syntax: `prepend = "digits"`

Description: Defines digits added to the beginning of a number.

Type: String

Optionality: Optional

Allowed: Any combination of decimal digits, or a null string (`""`)

Notes:

- If the `remove` and `prepend` parameters are both used in the same number rule, "*prepend*" is added to the beginning of the number after the number has been modified by the `remove` parameter.
- The `prepend` parameter is an element of the `NumberRules` parameter array.

Example:
```
prepend = "0"
```

remove

Syntax: `remove = num`

Description: The number of digits stripped from the beginning of a number.

Type: Integer

Optionality: Required

Notes: The value of the `remove` parameter must be less than or equal to the value of the *min* (on page 48) parameter.

Example: `remove = 2`

resultNoa

Syntax: `resultNoa = noa`

Description: A nature of address (NOA) sent to the network.

Type: Integer

Optionality: Optional

Notes:
- A value is typically specified in demoralization rules
- This parameter is an element of the `NumberRules` parameter array

Example: `resultNoa = 4`

RedirectNumberMappings Parameters

The following parameters are used to map the redirect number. They are found within `RedirectNumberMappings = [ ]`.

This section in optional.

Example RedirectNumberMappings configuration

Here is an example `RedirectNumberMappings` section of the `DCAInstances` configuration.

```
RedirectNumberMappings = [
{
    prefix = "641234"
    destination = "oracle.com"
    type = "SIP_URI"
}
]
```

destination

Syntax: `destination = "address"`

Description: The destination address string.

Type: String

Optionality: Mandatory if the `RedirectNumberMappings` section is included.

Allowed: See *RFC 4006*

Default: N/A

Example: `destination = oracle.com`

prefix

Syntax: `prefix = pref`

Description: A prefix of the destination Routing Address in the connect.

Type: Number string

Optionality: Mandatory if the `RedirectNumberMappings` section is included.
Allowed: Digits
Default: N/A
Example: `prefix = 641234`

type

Syntax: `type = "type"`
Description: The destination's type.
Type: String
Optionality: Required if the `RedirectNumberMappings` section is included.
Allowed:
- IPv4
- IPv6
- URL
- SIP_URI

Default: N/A
Example: `type = "SIP_URI"`

Tracing Parameters

The following parameters are used for tracing activities. They are all found within the `Tracing = { }` statement.

Example Tracing configuration

Here is an example `Tracing` section of the `DCAInstances` configuration.

```
Tracing = {

    enabled = true

    OrigAddress = [
        "a.b.c.com.0064212",
        "a.b.c.com.0064213",
        "a.b.c.com.0064214"
    ]

    destinationAddressAvp = 1234

    DestAddress = [
        "a.b.c.com.0064213",
        "a.b.c.com.0064214"
    ]

    traceDebugLevel = "all"
}
```

destAddress

Syntax: `destAddress = ["addr", "addr"]`
Description: List of destination addresses that are to be traced.
Type: String array
Optionality: Optional
Allowed:
- Any valid addresses
- ""

Default: ""

Notes: "" = trace all known destination addresses.
destAddress is set to *Dest-Realm.Subscription-Id*.

Example values:
```
destAddress = [
 "a.b.c.com.0064213",
 "a.b.c.com.0064214"
]
```

destinationAddressAvp

Syntax: `destinationAddressAvp = avp`

Description: The AVP to use in destination address as *RFC 4006* does not specify this.

Type: Integer

Optionality: Optional

Notes: If not specified, `destinationAddress` is hard-coded to `0000`

Example: `destinationAddressAvp = 1234`

enabled

Syntax: `enabled = true|false`

Description: Switches tracing on or off.

Type: Boolean

Optionality: Optional

Allowed: true, false

Default: false

Notes: If false, then the parameters in the Tracing section are ignored.

Example: `enabled = false`

origAddress

Syntax: `origAddress = ["addr", "addr"]`

Description: List of originating addresses that are to be traced.

Type: String array

Optionality: Optional

Allowed:
- Any valid addresses
- ""

Default: ""

Notes: "" = trace all known originating addresses.
origAddress is set to <Origin-Realm>.<Subscription-Id.>

Example values:
```
origAddrress = [
 "a.b.c.com.0064212",
 "a.b.c.com.0064213",
 "a.b.c.com.0064214"
]
```

traceDebugLevel

Syntax: `traceDebugLevel = "level"`

Description: The debug level the tracing be at should.

Type: String

Optionality: Mandatory

Notes: This is a string, with comma separation in it. See traceDebugLevel in *ACS Technical Guide*.

Useful flags are cdaconfig, diameterControlAgent, cdaObjectCounts

Example: `traceDebugLevel = "all"`

DiameterServer Parameters

The following parameters are used for a Diameter Server. They are all found within `DiameterServer = { }`.

Example DiameterServer Configuration

Here is an example `DiameterServer` section in the `DCAInstances` configuration.

```
DiameterServer = {
    protocol = "both"
    sctpListenPort = "3868"
    tcpListenPort = "3868"
    tcpBindAddress = "192.168.1.1"
    sctpBindAddress = "192.168.1.2"
    Auth-Application-Id = [ 4, 34, 42 ]
    Acct-Application-Id = 21
    Vendor-Specific-Application-Identifier = [
        {
            Vendor-Id = 111
            Auth-Application-Id = 1234
        }
        {
            Vendor-Id = 111
            Acct-Application-Id = 4321
        }
    ]
    localOriginHost = "creditcontrol.realm3.oracle.com"
    localOriginRealm = "realm3.oracle.com"
    productName = "oracle-dca"
    vendorId = 16247
    Supported-Vendor-Id = [ 16247, 10415 ]
    duplicateTime = 240
    duplicateBytes = 31457280
    connectionTimeout = 30
    watchdogPeriod = 30
    inBufferSize = 16384
    outBufferSize = 16384
    sendOriginStateId = false
    sendQuotaThreshold = true
    thirtyTwoBitQuotaThresholds = true
    percentTimeQuotaThreshold = 80
    percentVolumeQotaThreshold = 80
    commitGrantedOnTerminate = false
    allowMultiServiceIdentifier = false
    finalGrantUnused = false
    sessionLimit = 0
    throttleLimitError = 3004
    overLimitError = 3004
    counterLogInterval = 0
    throttleThreshold = 100
    throttleInterval = 100
    sendCreditLimitReachedOnSessionEnd = false
    chargeOnSessionTimeout = true
    sendAbortOnSessionTimeout = true
```

```
            commitGrantedOnSessionTimeout = true
            sessionFallbackTcc = 3600
        } # End of DiameterServer section
```

Acct-Application-Id

syntax: `Acct-Application-Id = [IDs]`

Description: The Acct-Application-Id AVP values to include in the Capabilities Exchange message.

Type: Integer – Single value, or array

Optionality: Optional

Notes: This array may have one or more values, or no value.

If there is only one value, brackets are not required.

If neither `Auth-Application-Id`, nor `Acct-Application-Id` is specified, then Auth-Application-Id = 4.

Examples: `Acct-Application-Id = 21`

allowMultiServiceIdentifier

Syntax: `allowMultiServiceIdentifier = true|false`

Description: Whether or not more than one Service-Identifier is supported for MSSC with a single multiple service credit control AVP.

Type: Boolean

Optionality: Optional (default used if not set).

Allowed:

- true – More than one allowed. Triggers a service per Service-Identifier
- false – More than one not permitted. Only use the first Service_Identifier

Default: false

Notes:

Example: `allowMultiServiceIdentifier = true`

Auth-Application-Id

syntax: `Auth-Application-Id = [IDs]`

Description: The Auth-Application-Id AVP values to include in the Capabilities Exchange message.

Type: Integer – Single value, or array

Optionality: Optional (default used if not set)

Notes: For Credit control this is 4.

This array may have one or more values, or no value.

If there is only one value, brackets are not required.

The first Auth-Application-Id (or 4 if none) is placed in that AVP in the CCR messages also.

If neither `Auth-Application-Id`, nor `Acct-Application-Id` is specified, then Auth-Application-Id = 4.

Default: 4

Examples: `Auth-Application-Id = [ 4, 34, 42 ]`

or

`Auth-Application-Id = 4`

chargeOnSessionTimeout

Syntax: `chargeOnSessionTimeout = true|false`

Description: Indicates how DCA should manage a timeout with an access device (for example, GGSN).

Type: Boolean

Optionality: Optional (default used if not set).

Allowed: If set to true, DCA will attempt to finalize any sessions with ACS that are associated with the timed-out session -

and..

there is an outstanding Apply Charging	DCA will respond with an Apply Charging Report with either the Total Granted Units or Total Used Units depending on configuration.
a service's charging is via SMCB (armed to report oAnswer; no outstanding ACh)	we send ERBCSM(oAnswer) to ACS.

Default: false

Notes:

Example: `chargeOnSessionTimeout = true`

commitGrantedOnSessionTimeout

Syntax: `commitGrantedOnSessionTimeout = true|false`

Description: Indicates whether DCA should request that the Total Granted Units or the Total Used Units should be committed.

Type: Boolean

Optionality: Optional (default used if not set).

Allowed:

Default: false

Notes:

Example: `commitGrantedOnSessionTimeout = false`

commitGrantedOnTerminate

Syntax: `commitGrantedOnTerminate = true|false`

Description: Whether or not to commit granted funds on session terminate where the used units are not specified.

Type: Boolean

Optionality: Optional (default used if not set).

Allowed:

- true – Commit granted (that is, charges for granted units)
- false – Only commit reported used units (that is, does not charge)

Default: false

Notes:

Example: `commitGrantedOnTerminate = true`

connectionTimeout

Syntax: `connectionTimeout = seconds`

Description: How long to wait for a reply before considering there is a transport level problem

Type: Integer
Optionality: Mandatory
Allowed: Seconds
Default: 30
Example: `connectionTimeout = 30`

counterLogInterval

Syntax: `counterLogInterval = secs`
Description: The interval in seconds between sending request counts to the syslog file. Set to 0 (zero) if you do not want to log requests.
Type: Integer
Optionality: Optional (default used if not set).
Allowed:
Default: 600
Notes: This parameter is also used to control the frequency of notice messages that log the number of requests received, and the frequency of warning messages that log the number of throttled requests.
Example: `counterLogInterval = 0`

duplicateBytes

Syntax: `duplicateBytes = bytes`
Description: How many bytes to allocate to the duplicate detection buffer
Type: Integer
Optionality: Mandatory
Default: 31457280 (30 MB)
Example: `duplicateBytes = 31457280`

duplicateTime

Syntax: `duplicateTime = seconds`
Description: How long to hold End-to-End Identifiers, when considering them for potential duplicates.
Type: Integer
Optionality: Mandatory
Allowed: Seconds
Default: 240
Example: `duplicateTime = 240`

finalGrantUnused

Syntax: `finalGrantUnused = true|false`
Description: Whether or not to allow re-granting of unused units after a final unit indication is sent.
Type: Boolean
Optionality: Optional (default used if not set).
Allowed: true, false
Default: false

Notes:
Example: `finalGrantUnused = true`

inBufferSize

Syntax: `inBufferSize = size`
Description: The size, in bytes, of inbound transport buffer.
Type: Integer
Optionality: Mandatory
Allowed:
Default: 0 (kernel default)
Example: `inBufferSize =16384`

throttleLimitError

Syntax: `throttleLimitError = Int`
Description: The error code generated when a throttle limit is breached.
Type: Integer
Optionality: Optional (default used if not set)
Allowed:
Default: 3004 – Diameter too busy
Notes:
Example: `throttleLimitError = 5006`

overLimitError

Syntax: `overLimitError = int`
Description: Sets the error code to use in a throttle-generated CCA, and when rejecting a session because the memory or session limit has been exceeded.
Type: Integer
Optionality: Optional (default used if not set).
Allowed:
Default: 3004 – Diameter too busy
Notes:
Example: `overLimitError = 3004`

protocol

Syntax: `protocol = "protocol"`
Description: The protocol for this server.
Type: String
Optionality: Mandatory
Allowed:
- "sctp"
- "tcp"
- "both"

Default: "tcp"
Example: `protocol = "tcp"`

sctpBindAddress

Syntax: `sctpBindAddress = "addr"`
Description: The SCTP port to listen on for this instance.
Type: String
Optionality: Mandatory
Default: 0 (that is, INADR_ANY)
Example: `sctpBindAddress = "192.168.1.2"`

sctpListenPort

Syntax: `sctpListenPort = "port"`
Description: The SCTP port to listen on
Type: String
Optionality: Mandatory
Default: "3868"
Example: `sctpListenPort = "3868"`

sessionLimit

Syntax: `sessionLimit = int`
Description: Limits the number of credit control sessions that may be created to the specified value.
Type: Integer
Optionality: Optional (default used if not set).
Allowed: A positive value.
Default: 0 – Do not apply a limit.
Notes:
Example: `sessionLimit = 0`

tcpBindAddress

Syntax: `tcpBindAddress = "addr"`
Description: The TCP port to listen on for this instance.
Type: String
Optionality: Mandatory
Default: 0 (that is, INADR_ANY)
Example: `tcpBindAddress = "192.168.1.1"`

tcpListenPort

Syntax: `tcpListenPort = "port"`
Description: The TCP port to listen on
Type: String
Optionality: Mandatory
Default: 3868
Example: `tcpListenPort = "3868"`

throttleThreshold

Syntax: `throttleThreshold = int`

Description: The number of initial or event requests to allow in a single interval. You set the length of the interval by using the `throttleInterval` parameter. The control agent counts the number of initial reservations or events received in the current interval and rejects new requests once the count has gone above the threshold.

Type: Integer

Optionality: Optional (default used if not set)

Allowed:

Default: 0 – Allow all requests

Notes:

Example: `throttleThreshold = 50`

throttleInterval

Syntax: `throttleInterval = int`

Description: The length, in milli-seconds, of each interval for which new requests will be counted and checked against the threshold specified in `throttleThreshold`.

Type: Integer

Optionality: Optional (default used if not set)

Allowed: None

Default: 100

Notes: If the value of the `throttleInterval` is set to any value other than `0` (zero), DCA rejects new requests and reports an error until the time set by the `throttleInterval`.

Example: `throttleInterval = 100`

localOriginHost

Syntax: `localOriginHost = "hostname"`

Description: The Origin-Host for messages sent out

Type: String

Optionality: Optional

Default: *"hostname"*

Notes: Recommended to keep the default value as the hostname of the target node, for example the SLC.

Example: `localOriginHost = "creditcontrol.realm3.oracle.com"`

localOriginRealm

Syntax: `localOriginRealm = "realmname"`

Description: The Origin-Realm for messages sent out

Type: String

Optionality: Mandatory

Notes: Each realm may contain at most one SLC

Example: `localOriginRealm = "realm3.oracle.com"`

outBufferSize

Syntax: `outBufferSize = size`
Description: The size, in bytes, of inbound transport buffer.
Type: Integer
Optionality: Mandatory
Allowed: Bytes
Default: 0 (kernel default)
Example: `outBufferSize = 16384`

percentTimeQuotaThreshold

Syntax: `percentTimeQuotaThreshold = percent`
Description: The percentage of granted service units of the time quota threshold.
Type: Integer
Optionality: Optional (default used if not set).
Allowed: 0 to 100 per cent
Default: 0
Notes:
Example: `percentTimeQuotaThreshold = 80`

percentVolumeQuotaThreshold

Syntax: `percentVolumeQuotaThreshold = percent`
Description: The percentage of granted service units of the volume quota threshold.
Type: Integer
Optionality: Optional (default used if not set).
Allowed: 0 to 100 percent
Default: 0
Notes:
Example: `percentVolumeQuotaThreshold = 80`

productName

Syntax: `productName = "name"`
Description: The product name used in Capabilities-Exchange-Answer
Type: String
Optionality: Mandatory
Allowed:
Default: "esg-dca" (that is, Diameter Control Agent)
Notes:
Example: `productName = "esg-dca"`

sendAbortOnSessionTimeout

Syntax: `sendAbortOnSessionTimeout = true|false`
Description: Indicates whether DCA will send an abort session request to the access device when the session with that device times out.
Type: Boolean
Optionality: Optional (default used if not set).

Allowed:
Default: false
Notes:
Example: `sendAbortOnSessionTimeout = false`

sendCreditLimitReachedOnSessionEnd

Syntax: `sendCreditLimitReachedOnSessionEnd = true|false`
Description: Indicates if we should reply to the final update request for a service (which DCA sometimes referred to as a sub-session) with a result code of DIAMETER_CREDIT_LIMIT_REACHED (4012), rather than DIAMETER_SUCCESS (2001).
Type: Boolean
Optionality: Optional (default used if not set).
Allowed:
Default:
Notes: For the record, the last update request for a service contains an MSCC with used service units but no requested-service-units AVP.
Example: `sendCreditLimitReachedOnSessionEnd = false`

sendOriginStateId

Syntax: `sendOriginStateId = true|false`
Description: To send or not send the origin state id flag.
Type: Boolean
Optionality: Optional (default used if not set).
Allowed: true, false
Default: true
Notes: Must be set to `false` if you do not want to send
Example: `sendOriginStateId = false`

sendQuotaThreshold

Syntax: `sendQuotaThreshold = true|false`
Description: Whether or not to send volume and quota threshold for MSCC.
Type: Boolean
Optionality: Optional (default used if not set).
Allowed: true, false
Default: true
Notes: Must be set to `false` if you do not want to send volume and quota threshold.
Example: `sendQuotaThreshold = false`

sessionFallbackTcc

Syntax: `sessionFallbackTcc = seconds`
Description: The session fallback tcc timer (in seconds).
Type: Integer
Optionality: Optional (default used if not set).
Allowed:
Default: 3600

Notes: This value is used as the tcc timer for sessions that do not have an associated service.

Example: `sessionFallbackTcc = 3600`

Supported-Vendor-Id

Syntax: `Supported-Vendor-Id = [ values ]`

Description: The Supported-Vendor-Id AVP values to include in the Capabilities Exchange message.

Type: Integer – Single value,or array

Optionality: Optional

Allowed:

Default:

Notes: This field may be specified as an array with either one or more values, or no value.

If there is only one value, brackets are not required.

Example:
```
Supported-Vendor-Id = [ 16247, 10415 ]
```

thirtyTwoBitQuotaThresholds

Syntax: `thirtyTwoBitQuotaThresholds = true|false`

Description: Whether to send the Time-Quota-Threshold and Volume-Quota-Threshold AVPs as 32-bit integers (as supported by a Cisco Release 9 GGSN).

Type: Boolean

Optionality: Optional (default used if not set).

Allowed: true, false

Default: true

Notes: If this is `false` it is sent as a 64-bit integer(as supported by a Cisco Release 7 GGSN).

Example:
```
thirtyTwoBitQuotaThresholds = true
```

Vendor-Specific-Application-Identifier

Syntax: `Vendor-Specific-Application-Identifier = [values]`

Description: The Vendor-Specific-Application-Id AVP values

Type: Array

Optionality: Optional

Allowed:

Default:

Notes:

Example:
```
Vendor-Specific-Application-Identifier = [
{
 Vendor-Id = 111
 Auth-Application-Id = 1234
}
{
 Vendor-Id = 111
 Auth-Application-Id = 4321
}
```

vendorId

Syntax: vendorId = *id*
Description: The Vendor ID to be supplied in the Capabilities-Exchange-Answer.
Type: Integer
Optionality: Mandatory
Allowed: A valid ID
Default: 0
Notes:
Example: vendorId = 16247

watchdogPeriod

Syntax: watchdogPeriod = *seconds*
Description: The period between sending out Device Watchdog messages to next-hop peer.
Type: Integer
Optionality: Mandatory
Allowed: Seconds
Default: 30
Example: watchdogPeriod =30

Services Configuration

Introduction

The Services section of the DCAInstances configuration allows you to select a specified service based upon one of the following:

- Service Context ID, Service Identifier, Rating Group, UnitType
- Configurable list of AVP values matched against the inbound message

If the SelectionAVPs is specified, then the serviceContextId, serviceIdentifier, ratingGroup, and unitType parameters cannot be specified.

Note: The service selection rules are tried in order. If you want the service configured by configurable AVPs to be used first, then put the parameters first in the Services section of the **eserv.config** file.

Services configuration structure

Here is the high-level structure of Services configuration section of the DIAMETER configuration in the **eserv.config** file.

```
Services = [
{
    serviceName = "<Service name>"

    serviceContextId = "<Service-Context-Id>"

    serviceIdentifier = "<Service-Identifier>"

    ratingGroup = "<Rating-Group>"

    screeningService = false

    unitType = "Time"
```

```
        selectionAVPsIsChargingKey = false

        conversionFactor = 1.0

        requestedAction="DIRECT_DEBITING"

        sleeServiceKey = 1231
        inapServiceKey = 1231
        tcc = 3600
        gracefulTerminationValidityTime= 300
        itc = "udi"
        validityTime = 30
        sleeTimeout = 10

        SelectionAVPs = [
            {
                AvpCodes = [ { avpCode=5, vendorId=16247 },
                             { avpCode=7, vendorId=16247 } ],
                avpType = "Unsigned64"
                avpValue = 123
                avpValue = "-0x5000000000"
                avpValue = "This Really Is A String"
                isChargingKey = false
            }
        ]

        AVPMappings = [
            {
                base_avpmappings
            }

            {
                basic_arrayavpmappings
            }

            {
                key_array_avpmappings
            }

            {
                array_with_conditions_avpmappings
            }

            {
                array_with_context_avpmappings
            }

            {
                conditional_avp_avpmappings
            }

            {
                prefix_tree_avpmappings
            }

            {
                timestamp_avpmappings
            }
        }
    }
```

See *Service Specific AVP Mappings* (on page 85) for a description of the `AVPMappings` configuration and examples of specific configuration and mappings.

Services parameters

The following parameters are used for a service. They are all located within the `Services` array.

As many services as required can be set up.

AvpCodes

Syntax: `AvpCodes = [avp_codes]`

Description: For a description of the `AVPCodes` parameters, see AvpCodes in the `DCADefaults` section.

Type: Array

Optionality: Optional

Allowed:

Default:

Notes: AvpCodes may be included as a member of the following:

- DCADefaults
- `SelectionAVPs` (on page 68)
- `typeCriteria` (on page 77)
- the base *Service Specific AVP Mappings* (on page 85)
- within AVP mappings of each level of a nested array

Example:

```
AvpCodes = [
{
 avpCode = 1234
 mandatory = true
 vendorId = "16747"
}
]
```

avpMappings

Syntax: `avpMappings = [avpMappings_parameters]`

Description: The service-specific AVP mappings.

Optionality: Optional

Default: If not present, will use the `avpMappings` in the `DCADefaults` section.

Notes: See *Service Specific AVP Mappings* (on page 85).

avpType

Syntax: `avpType = "type"`

Description: The AVP datatype to match.

Type: String

Optionality: Optional

Allowed: Allowed values are:

- Integer32
- Integer64
- Unsigned32
- Unsigned64
- UTF8String
- Enumerated

Default:

Notes: This is a member of the `SelectionAVPs` array.

Example:
```
paraMeter = "Integer32"
```

avpValue

Syntax: `avpValue = "value"`

Description: The value to match.

Type: It may be specified either as a number or a quoted string.

Optionality: Optional

Allowed:

Default:

Notes: The config file supports only the signed 32-bit range for numbers. For numbers outside of that range, put it in quotes.

This is a member of the `SelectionAVPs` array.

Example:
```
avpValue = 123
avpValue = "-0x5000000000"
avpValue = "This Really Is A String"
```

conversionFactor

Syntax: `conversionFactor = unit`

Description: The conversion factor to use when communicating with ACS.

- Multiplies the value received from ACS by this factor to calculate the Granted-Service-Unit AVP.
- Divides the Used-Service-Unit AVP by this value before sending it to ACS.

Type: Float

Optionality: Mandatory

Allowed:

Default: Defaults to:

- 0.1 for time
- 1048576 for octets
- 1.0 for everything else

Notes: Ignored for CC-Time AVPs (always uses the default of 0.1)

Example:
```
conversionFactor = 1.0
```

gracefulTerminationValidityTime

Syntax: `gracefulTerminationValidityTime = seconds`

Description: The number of seconds granted for the user to top up the account during graceful termination. Refer to *RFC 4006* A.7.

Type: Integer

Optionality: Optional

Allowed: in seconds

Default:

Notes: Not present means no graceful termination.

Example:
```
gracefulTerminationValidityTime = 300
```

inapServiceKey

Syntax: `inapServiceKey = value`

Description: The INAP Key value

Type: Integer

Optionality: Optional

Allowed: Any 32 bit integer

Example: `inapServiceKey = 1234`

isChargingKey

Syntax: `isChargingKey = true|false`

Description:

Type: Boolean

Optionality: Optional (default used if not set).

Allowed: true, false

Default:

Notes: If true, then we may match a Diameter CCR INITIAL_REQUEST, or UPDATE_REQUEST.

If false, then we may only match the INITIAL_REQUEST (or if that is empty, then the first UPDATE_REQUEST).

This is a member of the `SelectionAVPs` array.

Example: `isChargingKey = false`

itc

Syntax: `itc = infoTransferCapability`

Description: The Bearer Capability Information Element (Q.931 section 4.5.5) contains an Information Transfer Capability (ITC) field that is set automatically by DCA when DCA triggers ACS.

This parameter overrides the ITC value within the Bearer Capability Information Element.

Type: Integer or string

Optionality: Optional

Allowed: Allowed values are:

String	Integer	Hex	Description
"speech"	0	0x00	Speech
"udi"	8	0x08	Unrestricted Digital Information
"rdi"	9	0x09	Restricted Digital Information
"3.1kHzAudio"	16	0x10	3.1 kHz audio
"udiTA" or "7kHzAudio"	17	0x11	Unrestricted Digital Information with tones/ announcements
"video"	24	0x18	Video

Default: Defaults to one of the following:

- "speech" (0x00), if the Requested-Service-Unit AVP is set to CC-Time.
- "udi" (0x08), if otherwise.

Notes: If automatic setting of ITC is required, then this parameter should be absent.

Example:
```
itc = 16
```
or
```
itc = "3.1kHzAudio"
```
or
```
itc = 0x10
```

ratingGroup

Syntax: `ratingGroup = "number"`

Description: The number used to identify the rating group as part of a service triggering rule.

Type: String

Optionality: Optional

Allowed: The value in quotes must be a number.

Default:

Notes: This parameter must not be specified if `SelectionAVPs` is specified.

Example:
```
ratingGroup = "2"
```

requestedAction

Syntax: `requestedAction = "action"`

Description: The action performed by the service

Type: String

Optionality: Optional

Allowed:
- DIRECT_DEBITING
- REFUND_ACCOUNT
- CHECK_BALANCE
- PRICE_ENQUIRY

Default: not present

Notes: Not present indicates this service is for session based transactions.
See *RFC 4006* Requested-Action AVP.

Example:
```
requestedAction = "DIRECT_DEBITING"
```

screeningService

Syntax: `screeningService = true|false`

Description: If a screening service is not found for a particular service context ID, then DCA assumes that no screening needs to take place, that is, the session is allowed.

Type: Boolean

Optionality: Optional (default used if not set).

Allowed: true, false

Default: false

Notes: If the `screeningService` flag is true, then `serviceIdentifier` and `ratingGroup` should be blank.

Example:
```
screeningService = false
```

SelectionAVPs

Syntax: `SelectionAVPs = [selection_avps]`

Description: Specifies the AVPs to be matched in an incoming request for the service to be triggered.

Type: Array

Optionality: Optional

Allowed:

Default:

Notes: All of the selection AVPs must be matched in an incoming request for the service to be triggered.

If `SelectionAVPs` is specified then none of `serviceContextId`, `serviceIdentifier`, `ratingGroup`, or `unitType` can be specified.

Example:

```
SelectionAVPs = [
 {
 AvpCodes = [
 { avpCode=5, vendorId=16247 },
 { avpCode=7, vendorId=16247 } ],
 avpType = "Unsigned64"
 avpValue = 123
 avpValue = "-0x5000000000"
 avpValue ="This Really Is A String"
 isChargingKey = false
 }
]
```

selectionAVPsIsChargingKey

Syntax: `selectionAVPsIsChargingKey = true|false`

Description: Indicates whether this AVP is a charging key.

Type: Boolean

Optionality: Optional (default used if not set).

Allowed:

- true – Incoming UPDATE_REQUESTS messages matching this rule, for which there is no current session, will start a new session.
- false – Such messages will be rejected with a Diameter answer with an error result code.

Default: false

Notes:

Example: `selectionAVPsIsChargingKey = false`

serviceContextId

Syntax: `serviceContextId = "id"`

Description: The ID of the Service Context

Type: String

Optionality: Mandatory, if `SelectionAVPs` is *not* specified.

Notes: This parameter must not be specified if `SelectionAVPs` is specified.

You must specify both the `ServiceContextId` and `serviceIdentifier` to identify the service. See *RFC 4006*.

Example: `serviceContextId = "3"`

serviceIdentifier

Syntax: `serviceIdentifier = "Id"`

Description: The service identifier number.

Type: Number string

Optionality: Optional

Notes: This parameter must not be specified if `SelectionAVPs` is specified.

You must specify both the `ServiceContextId` and `serviceIdentifier` to identify the service. See *RFC 4006*.

Example: `serviceIdentifier = "3"`

serviceName

Syntax: `serviceName = "name"`

Description: The unique name of the service

Type: String

Optionality: Mandatory

Allowed: Any string

Default:

Example: `serviceName = "DirectDebitService"`

sleeServiceKey

Syntax: `sleeServiceKey = value`

Description: The Service Key value

Type: Integer

Optionality: Optional

Allowed: The value specified in the SERVICEKEY entry in the **SLEE.cfg** file.

For more information about the SERVICEKEY configuration, see *SLEE Technical Guide*

Default: no default

Example: `sleeServiceKey = 1234`

sleeTimeout

Syntax: `sleeTimeout = seconds`

Description: How long (in seconds) to wait for a response from the SLEE before the session times out

Type: Integer

Optionality: Optional (default used if not set)

Allowed:

Default: 10

Notes:

Example: `sleeTimeout = 15`

tcc

Syntax: `tcc = seconds`

Description: The session supervision timer timeout

Type: Integer

Optionality: Mandatory
Allowed: number of seconds
Default: 3600
Notes: Refer to *RFC 4006*
Example: `tcc = 3600`

unitType

Syntax: `unitType = "type"`
Description: The unit type used in the service
Type: String
Optionality: Mandatory, if `SelectionAVPs` is *not* specified.
Allowed:
- "Time"
- "Money"
- "Total-Octets"
- "Input-Octets"
- "Output-Octets"
- "Service-Specific"

Default: "Time"
Notes: This parameter must not be specified if `SelectionAVPs` is specified.
Example: `unitType = "Time"`

validityTime

Syntax: `validityTime = seconds`
Description: The validity time in seconds of granted units. Results in Validity-Time AVP being placed in CCA.
Type: Integer
Optionality: Optional (default used if not set).
Allowed:
Default: -1 (Not included)
Notes:
Example: `validityTime = 30`

Service Specific AVP Mappings parameters

The `AVPMappings` configuration in the `Services` section contains the following parameters that are used only in the `Services` section, not in other sections of the DIAMETER configuration.

For AVP parameters used throughout the DIAMETER configuration see *AvpMappings Parameters* (on page 32).

contextAVP

Syntax: `contextAVP = true|false`
Description: Defines whether the `avpCode` in the specified `AVPs` array is the context AVP to use in an *Array with Context* (on page 103)
Type: Boolean
Optionality: Optional.
Allowed:
Default:

Notes: When only a single unique AVP is used to establish context, that AVP is typically the key AVP associated with a data record. However DCA also allows more than 1 sub-AVPs in a hierarchy to be marked for inclusion for context. These multiple AVPs which form the context are know as the Context AVP.

Context AVPs are typically used when possible key values are not well known, or unique, or the key might otherwise rely on multiple items from the hierarchy.

Example:
```
contextAVP = true
```

encodedExtension

Syntax:
```
encodedExtension= {profile_parameters}
```

Description: This identifies the target tag and type in an incoming extension profile block for this AVP, when `extensionFormat = "encoded"`.

Type: Parameter list

Optionality: Optional

Allowed:

Default:

Notes: If `encodedExtension` is present and `extensionFormat` is absent, `extensionFormat = "encoded"` is assumed. See *extensionFormat* (on page 35) for details.

RAW_DATA profile mappings also have extra options (`octetsStart` and `octetsLength)` for specifying a part of the AVP (for inbound) or profile field (for outbound) to extract.

Example:
```
encodedExtension = {
 profileTag = 99123
 profileFormat = "INTEGER"
 octetsStart = 3
 octetsLength = 0
}
```

excludeIfMatches

Syntax:
```
excludeIfMatches = [ avpvalue ]
```

Description: The value of the AVPs to exclude from the Type Criteria matching.

Type: Array

Optionality: Optional (default used if not set).

Allowed: Integer, string, hex string

Default:

Notes: This parameter can be used as part of the Base mapping outside `AVPs = [ ... ]`.

If used inside `AVPs = [...]` it must be used as part of the `typeCriteria` section. See *typeCriteria* (on page 77) for an example.

Example: **Example 1**

```
excludeIfMatches = [ 101, 105 ]
```

Example 2

```
excludeIfMatches = [“SPAM”]
```

inapField

Syntax: `inapField = [ field1, field2, ... ]`

Description: Identifies the:
- Target INAP field(s) for mapping from this AVP for Inbound Mapping
- Source INAP field(s) for mapping to this AVP for Outbound Mapping

Type: String array

Optionality: Optional

Allowed: The following INAP fields are allowed:
- "additionalCallingpartyId"
- "calledPartyBcdNumber"
- "calledPartyNumber"
- "callingPartyNumber"
- "cause"
- "destinationRoutingAddress"
- "imsi"
- "locationInformation" - see note below
- "locationNumber"
- "maxCallDuration"
- "mscAddress"
- "originalCalledPartyId"
- "redirectingPartyId"
- "timeIfNoTariffSwitch"

Default:

Notes: If the AVP mappings are to and from INAP Field(s), please do not configure or specify parameters associated with acsProfile mapping (that is, `extensionFormat` should not be set to "encoded". `encodedExtension` should be absent).

The location information in the AVP is an encoded field. In ACS the location information is split up, to populate the call context buffers of MCC, MNC, LAC, and Cell ID, for originating and terminating. Refer to the *ACS Buffers* topic in *ACS Feature Nodes User's Guide*.

Example: `inapField= ["CalledPartyNumber"]`

includeIfMatches

Syntax: `includeIfMatches = [ avpvalue ]`

Description: The value of the AVPs to include in the Type Criteria matching.

Type: Array

Optionality: Optional (default used if not set).

Allowed: Integer, string, hex string

Default:

Notes: This parameter can be used as part of the Base mapping outside `AVPs = [ ... ]`.

If used inside `AVPs = [...]` it must be used as part of the `typeCriteria` section. See *typeCriteria* (on page 77) for an example.

Example: **Example 1**

```
includeIfMatches = [ 1, 10, 101, 1001, 10001 ]
```

Example 2

```
includeIfMatches = ["GoodNews!", "PrettyGoodNews"]
```

keyArray

Syntax: `keyArray = true|false`

Description: Defines whether the `avpCode` in the specified `AVPs` array is the key to use in a *Key Array* (on page 90).

Type: Boolean

Optionality: Optional

Allowed: true, false

Default:

Notes:

Example: `keyArray = true`

literal

Syntax: `literal = "value"`

Description: Applies the literal value to the AVP when the outbound message matches the types defined for that mapping.

Type: String

Optionality: Optional

Default:

Notes: Outbound AVP only.

If a mapping specifies both a literal and an `IncludesIfMatches` conditional AVP, then the literal will override the mapped value if the original value is found in the `IncludesIfMatches` array.

Example: `literal = "1"`

profileFormat

Syntax: `profileFormat = "format"`

Description: The format of the profile.

Type: String

Optionality: Optional

Allowed: The value given for this must be one of the valid storage formats for ACS profile fields. The allowable values for this parameter are:

- INTEGER
- INTEGER64
- UINTEGER
- UINTEGER64
- LNSTRING
- NSTRING
- STRING
- TIME
- BOOLEAN

- ARRAY
- RAW_DATA

Default: INTEGER

Notes: See also the related parameter, `profileTag`.

Part of `encodedExtension`.

If the `avpFormat` parameter is set to `"Grouped"` for the `AVPs` array, then `profileFormat` must be "ARRAY"

Example:
```
profileFormat = "LNSTRING"
```

profileTag

Syntax:
```
profileTag = num
```

Description: The profile tag.

Type: Integer

Optionality: Optional

Notes: This parameter is used to identify the profile tag it will be stored into/retrieved from. See also the related parameter, `profileFormat`.

Part of `encodedExtension`.

Example:
```
profileTag = 999
```

mappingTypes

Syntax:
```
mappingTypes = ["mapping_types"]
```

Description: Defines the message types between DCA and ACS that the mapping applies to.

Type: String Array

Optionality: Optional (default used if not set).

Allowed: For Inbound:

- "InitialRequest"
- "UpdateRequest"
- "EventRequest"
- "TerminateRequest"

For Outbound:

- "InitialResponse"
- "UpdateResponse"
- "EventResponse"
- "TerminateResponse"
- "FreeCallResponse" – See Notes.

One or more mapping types may be specified.

Note that the Configuration mappingType does not directly correspond to the CC-Request-Type. See *Mapping categories* (on page 75).

Default:
```
mappingTypes = ["InitialRequest", "InitialResponse",
"EventRequest", "EventResponse"]
```

Notes: If the configuration, within the `AVPs` array, for inbound is the same as for outbound, include the inbound and outbound message types in the list.

Each inbound configuration "Request" `mappingType` has a counterpart outbound "Response" `mappingType` that (when defined) is applied to the outbound Diameter message.

For example: If an inbound Diameter message has InitialRequest mappings applied, then InitialResponse mappings will be applied to the corresponding outbound Diameter answer. However, if a call is determined to be free, say after screening, or become free mid-session, then any mappings classified as "FreeCallResponse" (for the selected service) will be applied to the outbound Diameter answer instead of the default response mapping type.

Example:
```
mappingTypes = ["InitialRequest", "UpdateRequest",
"EventRequest", "TerminateRequest"]
```

Mapping categories

This table describes the relationship between CC-Request-Type and Configuration `mappingTypes` for outbound request type mappings:

Diameter Message	CC-Request-Type	Credit Control	Service	Requested-Service-Unit	Used-Service-Unit	Configuration mappingTypes
CCR	INITIAL_REQ	MSCC	New	New	-	InitialRequest
CCR	INITIAL_REQ	Basic	New	New	-	InitialRequest
CCR	UPDATE_REQ	MSCC	New	New	-	InitialRequest
CCR	UPDATE_REQ	MSCC	Existing	Existing UT	Existing UT	UpdateRequest
CCR	UPDATE_REQ	Basic	Existing	Existing UT	Existing UT	UpdateRequest
CCR	UPDATE_REQ	MSCC	Existing	-	Existing UT	TerminateRequest
CCR	TERM_REQ	MSCC	any	-	-	TerminateRequest
CCR	TERM_REQ	Basic	any	-	-	TerminateRequest
CCR	EVENT_REQ	-	-	-	-	EventRequest

Here are the abbreviations used in the table.

Abbreviation	Description
CCR	Credit-Control-Request
INITIAL_REQ	INITIAL_REQUEST
UPDATE_REQ	UPDATE_REQUEST
TERM_REQ	TERMINATION_REQUEST
EVENT_REQ	EVENT_REQUEST
MSCC	Multiple-Services-Credit-Control
Basic	Basic Credit-Control
Existing UT	Existing Unit Type

octetsLength

Syntax: `octetsLength = num`

Description: The number of octets to extract from the source data.

Type: Integer
Optionality: Optional (default used if not set).
Allowed:
Default: 0 (copy all octets from `octetsStart` until the end of the field)
Notes: Part of `encodedExtension`.
Used if `profileFormat` is RAW_DATA.
Used in conjunction with `octetsStart`.
Example: See examples in `octetsStart`.

octetsStart

Syntax: `octetsStart = num`
Description: The offset within the source data (AVP or profile field) to start copying from.
Type: Integer
Optionality: Optional (default used if not set).
Allowed:
Default: 0 (start copying from the beginning)
Notes: Negative values can be used to specify an offset from the end of the data.
Part of `encodedExtension`.
Used if `profileFormat` is RAW_DATA.
Used in conjunction with `octetsLength`.
Example: Example 1: Copy all data except for the first 3 octets.

```
octetsStart = 3
octetsLength = 0
```

Example 2: Copy the third-to-last and second-to-last octets.

```
octetsStart = -3
octetsLength = 2
```

repeating

Syntax: `repeating = true|false`
Description: Specifies whether the `avpCode` is repeating, that is the Diameter message contains an array-like structure where the `avpCode` is used more than once.
Type: Boolean
Optionality: Optional (default used if not set).
Allowed: true, false
Default: false
Notes: For inbound mappings, the destination will need to be an Array, or Prefix Tree type profile tag, in order to handle the complex structure.
Outbound mappings need to come from an Array or Prefix Tree.
Example: `repeating = true`

timestamp

Syntax: `timestamp = "timestamp"`
Description: The timestamp to record
Type: String
Optionality: Optional

Allowed: The available timestamps are:

- "TIME_REQUEST_RECEIVED32"
- "TIME_REQUEST_RECEIVED64"
- "TIME_NOW32"
- "TIME_NOW64"

Default:

Notes: 'NOW' refers to the time the mapping is processed. For an outbound mapping, it will be as the reply is being created, hopefully immediately before it is sent.

'REQUEST_RECEIVED' is the time the request [that we are processing] entered the Diameter stack.

See *Timestamp* (on page 118) for example usage.

Example: `timestamp = "TIME_REQUEST_RECEIVED32"`

typeCriteria

Syntax: `typeCriteria = [criteria]`

Description: Lists the criteria to match on, then specifies the AVP that is searched and, if a match is found, mapped to a profile tag.

Type: Parameter section

Optionality: Optional (default used if not set).

Allowed:

Default:

Notes:

Example: In this example, if the value of AVP of 1000.2000.3000 is 1, then map the value of each of the AVPs 1000.2000.3001 to array element as tag 100 in the sub-profile block, as a string.

Note, the parent AvpCodes 1000.2000 are not shown in this example configuration fragment.

```
typeCriteria = [
 {
 includeIfMatches = [ 1 ]
 AvpCodes = [
 { avpCode = 3000
 }
 ]
 avpFormat = "Integer32"
 # What is included if match found
 AVPs = [
 {
 AvpCodes = [
 { avpCode = 3001
 repeating = true
 }
 ]
 avpFormat = "UTF8String"
 encodedExtension = {
 profileTag = 100
 profileFormat = "STRING"
 }
 }
 ]
 }
]
```

See the example mapping for this configuration in *Mapping* (on page 96).

See more examples in:

- *Array with Conditions* (on page 93)
- *Conditional AVP* (on page 108)

PeerSchemes Configuration Section

PeerSchemes configuration structure

Here is a high level structure of the configuration of a scheme in the `PeerSchemes` section.

```
PeerSchemes = [
{
    schemeName = "SchemeA"

    Peers = [
    {
        name = "host1"

        protocol = "both"

        permittedOriginHosts = [
            "host1.realm1.oracle.com"
        ]
```

```
            RemoteAddresses = [
                "192.168.1.10"
            ]

            netmaskBits = 32

            netmask6Bits = 128

            permittedInstances = 0

            reqSctpInboundStreams = 8
            reqSctpOutboundStreams = 8
        } # end of Peer host1

        {
            peer_host2_parameters
        } # end of Peer host2

        ]
    } # End of Scheme A

    {
        schemeName = "SchemeB"

            SchemeB_parameters

        } # End of Scheme B
    ] # End of PeerSchemes section
```

Note: Default settings are specified at installation time.

PeerSchemes parameters

The following parameter is used in the `PeerSchemes` array.

schemeName

Syntax: `schemeName = "name"`
Description: The name identifying the scheme.
Type: String
Optionality: Mandatory
Example: `schemeName = "SchemeA"`

Peer host parameters

The following parameters are used for a peer host. They are members of the `Peers` array.

You can be set up as many peer hosts as required. A scheme can have no peers, in which case will accept all connections.

name

Syntax: `name = "name"`
Description: The name identifying either peer, or group of peers.
Type: String
Optionality: Mandatory
Example: `name = "host1"`

netmaskBits

Syntax: `netmaskBits = bits`
Description: The number of bits for netmask.
Type: Integer
Optionality: Mandatory
Default: 32 (bits for netmask, that is, a single machine (/32))
Example: `netmaskBits = 32`

netmask6Bits

Syntax: `netmask6Bits = bits`
Description: The number of bits for the IP version 6 prefix
Type: Integer
Optionality: Mandatory
Default: 128 (bits for the address prefix, that is, a single machine (/128))
Example: `netmask6Bits = 128`

permittedInstances

Syntax: `permittedInstances = num`
Description: The number of permitted instances.
Type: Integer
Optionality: Mandatory
Notes: If set to 0 then allow all.
Example: `permittedInstances = 0`

permittedOriginHosts

Syntax: `permittedOriginHosts = "host"`
Description: The list of peer names which will be checked against the OriginHost AVP, during the capabilities exchange.
Type: String
Optionality: Mandatory
Example value: `permittedOriginHosts = "host1.realm1.oracle.com"`

protocol

Syntax: `protocol = "protocol"`
Description: The protocol for this host peer.
Type: String
Optionality: Optional
Allowed:
- "sctp"
- "tcp"
- "both"

Default: If not specified, then it uses the protocol from the *DiameterServer section.* (on page 52)
Example: `protocol = "tcp"`

RemoteAddresses

Syntax: `remoteAddresses = ["ipaddress"]`

Description: The list of remote IP addresses.

Type: Array of string parameters

Optionality: Mandatory

Notes: If an address becomes unavailable the list will be cycled through.

Example:
```
remoteAddresses = [
"192.168.1.10"
]
```

reqSctpInboundStreams

Syntax: `reqSctpInboundStreams = num`

Description: The number of requested inbound sctp streams.

Type: Integer

Optionality: Mandatory

Notes: There is no guarantee you will actually get these.

Example: `reqSctpInboundStreams = 8`

reqSctpOutboundStreams

Syntax: `reqSctpOutboundStreams = num`

Description: The number of requested outbound sctp streams.

Type: Integer

Optionality: Mandatory

Notes: There is no guarantee you will actually get these.

Example: `reqSctpOutboundStreams = 8`

Example PeerSchemes section

Here is an example `PeerSchemes` section of the DIAMETER configuration in the **eserv.config** file.

```
PeerSchemes = [
{
    schemeName = "SchemeA"

    Peers = [

    {
        name = "host1"

        protocol = "both"

        permittedOriginHosts = [
            "host1.realm1.oracle.com"
        ]

        RemoteAddresses = [
            "192.168.1.10"
        ]

        netmaskBits = 32

        netmask6Bits = 128
```

```
            permittedInstances = 0

            reqSctpInboundStreams = 8
            reqSctpOutboundStreams = 8
        } # end of Peer host1

        {
            name = "host2"

            protocol = "sctp"

            permittedOriginHosts = [
                "host1.realm1.oracle.com"
            ]

            RemoteAddresses = [
                "192.168.1.11"
            ]

            netmaskBits = 32

            netmask6Bits = 128

            permittedInstances = 0

            reqSctpInboundStreams = 8
            reqSctpOutboundStreams = 8
        } # end of Peer host2
    } # End of Scheme A

    {
        schemeName = "SchemeB"

        Peers = [
        {
            name = "host1"

            protocol = "both"

            permittedOriginHosts = [
                "host1.realm1.oracle.com"
            ]

            RemoteAddresses = [
                "192.168.1.10"
            ]

            netmaskBits = 32

            netmask6Bits = 128

            permittedInstances = 0

            reqSctpInboundStreams = 8
            reqSctpOutboundStreams = 8
        } # end of Peer host1

    ]
    } # End of Scheme B
    ] # End of PeerSchemes section
```

Statistics Logged by diameterControlAgent

Introduction

Diameter statistics are generated by each SLC, and then transferred at periodic intervals to the Service Management System (SMS) for permanent storage and analysis.

An existing statistics system (smsStats) provides functions for the collection of basic statistical events. This is provided in the NCC SMS application. Refer to *SMS Technical Guide* for details.

DCA statistics

SMS statistics are logged with APPLICATION_ID = 'DCA' (application number 96)

The following statistics are defined:

- DUPLICATES_DETECTED
- INITIAL_REQUESTS_RECEIVED
- INITIAL_REQUESTS_ANSWERED
- UPDATE_REQUESTS_RECEIVED
- UPDATE_REQUESTS_REJECTED_ANSWERED
- TERMINATION_REQUESTS_RECEIVED
- TERMINATION_REQUESTS_ANSWERED
- DIRECT_DEBITS_RECEIVED
- DIRECT_DEBITS_ANSWERED
- ACCOUNT_REFUNDS_RECEIVED
- ACCOUNT_REFUNDS_ANSWERED
- BALANCE_CHECKS_RECEIVED
- BALANCE_CHECKS_ANSWERED
- PRICE_ENQUIRIES_RECEIVED
- PRICE_ENQUIRIES_ANSWERED
- UNSUPPORTED_MESSAGES
- SESSIONS_TIMED_OUT
- GENERIC_ACTION_RECEIVED (Tracks requests for non-standard triggering)
- GENERIC_ACTION_ANSWERED (Tracks answers for non-standard triggering)
- RAR_UNABLE_TO_BE_DELIVERED (RAR cannot deliver RAR to the Diameter client)
- RAR_ERRORS_RECEIVED (RAR received error response to RAR)
- RAR_TIMEOUT (RAR has timed out)
- RAR_SENT (RAR sent from DCA)
- RAR_ANSWERS_RECEIVED (RAA received in response to RAR)
- RAR_UNABLE_TO_COMPLY_RECIEVED (RAA received with UNABLE_TO_COMPLY)
- RAR_LIMITED_SUCCESS_RECEIVED (RAA received with LIMITED_SUCCESS)
- RAR_UNSOLICITED_ANSWER (Unexpected RAA received)
- RAA_EXCEPTION (RAA exception handling)
- RAA_UNKNOWN_SESSION_ID (RAA received with unknown session ID)

For all statistics, the Origin-Realm AVP from the message received is put into SMF_STATISTICS.DETAIL.

Reports

The following reports are available:

- DCA System Stats
- DCA System Stats by Realm

Reports are generated using the SMS Report Functions screen. Refer to the *SMS User's Guide* for details.

Example report

Here is an example DCA System Stats by Realm report.

```
DCA Statistics Listing by Realm
===============================
Start Date: 16 August 2007
Finish Date: 18 August 2007
Report Type: All Entries
Realm: realm2.oracle.com

28 August 2007, 22:50:56


Node Name          Statistics ID                  Date       Value
------------------ ------------------------------ ------------------- ----------
mtv-tst-scp10      DUPLICATES_DETECTED            17 August 07 00:52 1
mtv-tst-scp10      DUPLICATES_DETECTED            17 August 07 00:54 1
mtv-tst-scp10      INITIAL_REQUESTS_ANSWERED      16 August 07 00:02 1
mtv-tst-scp10      INITIAL_REQUESTS_ANSWERED      16 August 07 03:04 1
mtv-tst-scp10      INITIAL_REQUESTS_ANSWERED      16 August 07 22:34 1
mtv-tst-scp10      INITIAL_REQUESTS_ANSWERED       7 August 07 00:52 2
mtv-tst-scp10      INITIAL_REQUESTS_ANSWERED      17 August 07 00:54 2
mtv-tst-scp10      INITIAL_REQUESTS_ANSWERED      17 August 07 01:00 1
mtv-tst-scp10      INITIAL_REQUESTS_RECEIVED      16 August 07 00:02 1
mtv-tst-scp10      INITIAL_REQUESTS_RECEIVED      16 August 07 03:04 1
mtv-tst-scp10      INITIAL_REQUESTS_RECEIVED      16 August 07 22:34 1
mtv-tst-scp10      INITIAL_REQUESTS_RECEIVED      17 August 07 00:52 2
mtv-tst-scp10      INITIAL_REQUESTS_RECEIVED      17 August 07 00:54 2
mtv-tst-scp10      INITIAL_REQUESTS_RECEIVED      17 August 07 01:00 1
mtv-tst-scp10      SESSIONS_TIMED_OUT             17 August 07 00:54 1
mtv-tst-scp10      UPDATE_REQUESTS_ANSWERED       16 August 07 00:02 1
mtv-tst-scp10      UPDATE_REQUESTS_ANSWERED       16 August 07 03:04 1
mtv-tst-scp10      UPDATE_REQUESTS_ANSWERED       16 August 07 22:34 1
mtv-tst-scp10      UPDATE_REQUESTS_ANSWERED       17 August 07 00:52 2
mtv-tst-scp10      UPDATE_REQUESTS_ANSWERED       17 August 07 00:54 2
mtv-tst-scp10      UPDATE_REQUESTS_ANSWERED       17 August 07 01:00 1
mtv-tst-scp10      UPDATE_REQUESTS_RECEIVED       16 August 07 00:02 1
mtv-tst-scp10      UPDATE_REQUESTS_RECEIVED       16 August 07 03:04 1
mtv-tst-scp10      UPDATE_REQUESTS_RECEIVED       16 August 07 22:34 1
mtv-tst-scp10      UPDATE_REQUESTS_RECEIVED       17 August 07 00:52 2
mtv-tst-scp10      UPDATE_REQUESTS_RECEIVED       17 August 07 00:54 2
mtv-tst-scp10      UPDATE_REQUESTS_RECEIVED       17 August 07 01:00 1

Completed
```

Chapter 4

Service Specific AVP Mappings

Overview

Introduction

This chapter explains the structure of the AVP mappings for a service.

In this chapter

This chapter contains the following topics.

Introduction

Introduction

The AVP mappings within the `Services` parameter section are organized as shown in *Services configuration structure* (on page 62).

There are two types of configuration formats available:

- 'classic' format:
  - Base
- 'nested' format. These are configured within an array of format `AVPs = [ ]`
  - *Basic Array* (on page 87)
  - *Key Array* (on page 90)
  - *Array with Conditions* (on page 93)
  - *Array with Context* (on page 103)
  - *Conditional AVP* (on page 108)
  - *Prefix Tree* (on page 116)
  - *Timestamp* (on page 118)

Note: If you use classic format, you cannot use nested format in the **eserv.config** file.

Base example

Here is an example of the base AVP mappings in the Services `AVPMappings` section.

General Example 1 – Classic Format. Specify AVP code(s) for this AVP. There MUST be one specified for the base AVP, plus list all extras for grouped AVPs.

```
{
    AvpCodes = [
    {
        avpCode = 1234
        mandatory = true|false
        vendorId = "VendorID"
        }
    ]
    # The AVP data format.
    avpFormat =
    "OctetString|Integer32|Integer64|UInteger32|UInteger64|Unsigned32|Unsigned64|
    Address|Time|UTF8String|DiameterIdentity|DiameterURI|Enumerated|"
    extensionType = 1234
    extensionFormat = "inapnumber|asn1integer|octets|encoded"

    encodedExtension= {
        profileTag = 99123
        profileFormat = "INTEGER | INTEGER64 | UINTEGER | UINTEGER64 | LNSTRING |
        NSTRING | STRING | TIME | BOOLEAN | RAW_DATA"
        octetsStart = 3
        octetsLength = 0
    }
inapField = [ field1, field2, ... ]
}
```

Simple conditional

Here is an example using `includeIfMatches` within `Services AvpMappings` section, that is, outside an `AVPs` array in a *typeCriteria* (on page 77) array.

```
{
    AvpCodes = [
        {
            avpCode = 4700
        }
        {
            avpCode = 2000
        }
    ]
    avpFormat = "UTF8String"
    includeIfMatches = [ "Good News!", "Pretty Good News!", "Over the moon!" ]
    encodedExtension = {
        profileTag = 94701
        profileFormat = "STRING"
    }
}
```

Nested format

Nested formats are generally used to define arrays and conditional AVPs. Definitions are nested in the array formatted `AVPs = []` and mirror the hierarchy of a Grouped AVP.

As groups can contain sub-groups, defined "AVPs" can contain sub-"AVPs".

Example nested format

The following example shows an AVP nested up to four levels deep:

- A root-level AVPs with one entry
- A first-level AVPs with one entry
- A second-level AVPs with two entries
- A third-level AVPs with two entries, which are part of the first entry of the second level AVPs.

Here is the configuration structure in the `DIAMETER Services AvpMappings` array of the example **eserv.config**.

```
AVPs = [ # Root-Level AVPs
    { # 1st Entry of Root-Level AVPs
        AvpCodes = [ ... ]
        AVPs = [ # 1st-Level AVPs
            { # 1st Entry of 1st-Level AVPs
                AvpCodes = [ ... ]
                AVPs = [ # 2nd-Level AVPs
                    { # 1st Entry of 2nd-Level AVPs
                        AvpCodes = [ ... ]
                        AVPs = [ # 3rd-Level AVPs
                            { # 1st Entry of 3rd-Level AVPs
                                AvpCodes = [ ... ]
                                avpFormat = "..."
                                encodedExtension = {
                                 profileTag = 80301
                                 profileFormat = "..."
                                }
                            }
                            { # 2nd Entry of 3rd-Level AVPs
                                AvpCodes = [ ... ]
                                avpFormat = "..."
                                encodedExtension = {
                                 profileTag = 80303
                                 profileFormat = "..."
                                }
                            } # End of 2nd Entry of 3rd-Level AVPs
                        ] # End of 3rd-Level AVPs
                    } # End of 1st Entry of 2nd-Level AVPs
                    { # 2nd Entry of 2nd-Level AVPs (optional; specify if applicable)
                        AvpCodes = [ ... ]
                        avpFormat = "..."
                        encodedExtension = {
                            profileTag = 80302
                            profileFormat = "..."
                        }
                    } # End of 2nd Entry of 2nd-Level AVPs
                ] # End of 2nd-Level AVPs
                } # End of 1st Entry of 1st-Level AVPs
            ] # End of 1st-Level AVPs
        } # End of 1st Entry of Root-Level AVPs
    ] # End of Root-Level AVPs
avpFormat = "Grouped"
encodedExtension = {
    profileTag = 8234
    profileFormat = "ARRAY"
}
```

Basic Array

Introduction

Basic Arrays are also known as "simple repeating AVPs". In the simplest case, the repeating AVP is the one which requires mapping to an array in a profile block. DCA will need to establish multiple instances of the same AVP. However you only need define a single Basic Array type mapping definition. The mapping definition needs to establish:

- That the target (or source) profile field is an array

- The format of the elements in the target array (for example, STRING)
- The format of the AVP (such as UTF8String)
- That the AVP code is repeating, that is, `repeating = true`. For a definition, see *repeating* (on page 76).

Note that in this case (unlike Paired-AVPs or Array with Conditions), there is no key or sub-AVPs to consider.

Basic Array configuration

Here is the example basic array configuration in the Services `AVPMappings` section of the **eserv.config**. In this example, the basic array contains a list of string-type (that is, `profileFormat = "STRING"`) elements.

```
{
    AVPs = [ # Root-Level AVPs
        { # 1st Entry in Root-Level AVPs
            AvpCodes = [
                {
                    avpCode = 6000
                }
            ]
            AVPs = [ # 1st Level AVPs
                {
                    AvpCodes = [
                        {
                                avpCode = 1000
                        }
                    ]
                    AVPs = [ # 2nd Level AVPs
                        {
                            AvpCodes = [
                                {
                                    avpCode = 2000
                                    repeating = true
                                }
                            ]
                            avpFormat = "UTF8String"
                            encodedExtension = {
                                profileTag = 9998
                                profileFormat = "STRING"
                            }
                        }
                    ] # End of 2nd Level AVPs
                }
            ] # End of 1st Level AVPs
        } # End of 1st Entry in Root-Level AVPs
    ] # End of Root-Level AVPs

    # Specify mapping applies INBOUND only.
    mappingTypes = ["InitialRequest", "UpdateRequest", "EventRequest",
    "TerminateRequest"]

    avpFormat = "Grouped"
    encodedExtension = {
        profileTag = 8000
        profileFormat = "ARRAY"
    }
}
```

Note: The parameter setting of `avpFormat = "Grouped"` can only be used in this place in the config file, that is, *outside* an "`AVPs = [...]`" array. Also an "`AVPs = [...]`" array must be defined because "Grouped" makes all the AVPs defined inside the "`AVPs = [...]`" array as belonging to the one group. Because of this it makes no sense to put `avpFormat = "Grouped"` anywhere else except here.

Example Basic Array configuration

Here is a worked example of a basic array: inbound configuration.

```
{
    AVPs = [
        {
            AvpCodes = [
                { avpCode = 1000 }
            ]
            AVPs = [
                {
                    AvpCodes = [
                        { avpCode = 2000 # <-- 1000.2000
                         repeating = true # <-- 2000 is repeating
                        }
                    ]
                    avpFormat = "UTF8String"
                    encodedExtension {
                        profileTag = 9998 # Array element in sub-profile block
                        profileFormat = "STRING"
                    }
                }
            ]
        }
    ]

    # Specify mapping applies INBOUND only.
    mappingTypes = ["InitialRequest", "UpdateRequest", "EventRequest",
    "TerminateRequest"]

    # AVPs above are encoded into an ARRAY-type in an ACS Profile Block.
    encodedExtension = {
        profileTag = 8000
        profileFormat = "ARRAY"
    }
}
```

The configuration for basic arrays: outbound is identical to above, except the mapping types are:

```
    # Specify mapping applies OUTBOUND only.
    mappingTypes = ["InitialResponse", "UpdateResponse", "EventResponse",
    "TerminateResponse"]
```

Mapping

This table shows the mapping of Diameter AVPs to ACS profile blocks resulting from the worked example basic arrays configuration section above. This example is for inbound.

Diameter AVP				Profile Block			
Root AVP	**1st Level**	**Value**		**Profile (ARRAY)**	**Sub-Profile Block**	**Tag**	**Value**
1000	2000	"049772056"		8000	1	9998	"049772056"
	2000	"6449016000"			2	9998	"6449016000"
	2000	"0800500600"			3	9998	"0800500600"

The mapping is a simple 1:1 mapping. For outbound, the mappings are the same, but in the reverse direction.

Key Array

Introduction

A key array is a set of records where one of the elements within each record can be identified as a key for accessing the record, using the parameter setting `keyAVP = true`.

Key Arrays configuration

Here is the example Key Array configuration in the Services `AVPMappings` section of the **eserv.config**. In this example, elements of the key array contain:

- an integer type Key AVP (`avpCodes = 3000`)
- three Data AVPs.

```
{
    AVPs = [ # Root-Level AVPs
        { # 1st Entry in Root-Level AVPs
            AvpCodes = [
                {
                    avpCode = 6100
                }
            ]
            AVPs = [ # 1st-Level AVPs
                { # 1st Entry in 1st-Level AVPs
                    AvpCodes = [
                        {
                            avpCode = 1000
                        }
                    ]
                    AVPs = [ # 2nd-Level AVPs
                        { # 1st Entry of 2nd-Level AVPs
                            AvpCodes = [
                                {
                                  avpCode = 2000
                                  repeating = true
                                }
                            ]
                            AVPs = [ # 3rd-Level AVPs (Final Level)
                                { # Mapping for Key AVP
                                  AvpCodes = [
                                    {
                                      avpCode = 3000
                                    }
                                  ]
                                  keyAVP = true
                                  avpFormat = "Integer32"
                                  encodedExtension = {
                                     profileTag = 100
                                     profileFormat = "INTEGER"
                                  }
                                } # End of Mapping for Key AVP
                                { # Mapping for 1st Data AVP
                                  AvpCodes = [
                                     {
                                       avpCode = 3001
                                     }
```

```
                                    ]
                                    avpFormat = "UTF8String"
                                    encodedExtension = {
                                       profileTag = 101
                                       profileFormat = "STRING"
                                    }
                                  } # End of Mapping for 1st Data AVP
                                  { # Mapping for 2nd Data AVP
                                    AvpCodes = [
                                       {
                                         avpCode = 3002
                                       }
                                    ]
                                    avpFormat = "UTF8String"
                                    encodedExtension = {
                                       profileTag = 102
                                       profileFormat = "STRING"
                                    }
                                  } # End of Mapping for 2nd Data AVP
                                  { # Mapping for 3rd Data AVP
                                    AvpCodes = [
                                      {
                                        avpCode = 3003
                                      }
                                    ]
                                    avpFormat = "UTF8String"
                                    encodedExtension = {
                                       profileTag = 103
                                       profileFormat = "STRING"
                                    }
                                  } # End of Mapping for 3rd Data AVP
                               ] # End of 3rd-Level AVPs
                            } # End of 1st entry of 2nd-Level AVPs
                        ] # End of 2nd Level AVPs
                    } # End of 1st Entry in 1st-Level AVPs
                ] # End of 1st-Level AVPs
            } # End of 1st Entry in Root-Level AVPs
        ] # End of Root-Level AVPs Mappings

        # Specify mapping applies INBOUND only.
        mappingTypes = ["InitialRequest", "UpdateRequest", "EventRequest",
        "TerminateRequest"]

        avpFormat = "Grouped"
        encodedExtension = {
            profileTag = 8001
            profileFormat = "ARRAY"
        }
}
```

Example Key Arrays configuration

Here is a worked example of Key Arrays: Inbound configuration.

```
{
    AVPs = [
        {
            AvpCodes = [
                { avpCode = 1000 }
            ]
            AVPs = [
                {
                    AvpCodes = [
```

```
                        { avpCode = 4400            # <-- 1000.4400
                          repeating = true          # <--      4400 is repeating
                        }
                    ]
                    AVPs = [
                        {
                            AvpCodes = [
                                { avpCode = 4410    # <-- 1000.4400.4410
                                }
                            ]
                            keyAVP = true              # <--      4410 is the key,
                            others below are data AVPs.
                            avpFormat = "Integer32"
                            encodedExtension {
                                profileTag = 9998   # Array element in sub-profile
                                block
                                profileFormat = "INTEGER"
                            }
                        }
                        {
                            AvpCodes = [
                                { avpCode = 4420    # <-- 1000.4400.4420
                                }
                            ]
                            avpFormat = "UTF8String"
                            encodedExtension {
                                profileTag = 9999   # Array element in sub-profile
                                block
                                profileFormat = "STRING"
                            }
                        }
                        {
                            AvpCodes = [
                                { avpCode = 4410    # <-- 1000.4400.4430
                                }
                            ]
                            avpFormat = "UTF8String"
                            encodedExtension {
                                profileTag = 10000  # Array element in sub-profile
                                block
                                profileFormat = "STRING"
                            }
                        }
                    ]
                }
            ]
        }
    ]

    # Specify mapping applies INBOUND only.
    mappingTypes = ["InitialRequest", "UpdateRequest", "EventRequest",
    "TerminateRequest"]

    encodedExtension = {
        profileTag = 8000
        profileFormat = "ARRAY"
    }
}
```

The configuration for Key Arrays: Outbound is identical to above, except the mapping types are:

```
    # Specify mapping applies OUTBOUND only.
    mappingTypes = ["InitialResponse", "UpdateResponse", "EventResponse",
    "TerminateResponse"]
```

Mapping

This table shows the mapping of Diameter AVPs to ACS profile blocks resulting from the worked example key array configuration section above. This example is for inbound.

Diameter AVP				Profile Block			
Root AVP	1st Level	2nd Level	Value	Profile Block (ARRAY)	Sub-Profile Block	Tag	Value
1000	4400	4410	220	8000	1	9998	220
		4420	"6449016000"			9999	"6449016000"
		4430	"Chris"			10000	"Chris"
	4400	4410	221		2	9998	221
		4420	"123123123"			9999	"123123123"
		4430	"Fred"			10000	"Fred"

The mapping is a simple 1:1 mapping. For outbound, the mappings are the same, but in the reverse direction.

Array with Conditions

Introduction

In the case where the AVP to be mapped is the child of a repeating AVP, or where there are multiple mappings, to be made for different child AVPs, you may apply Array With Conditions mapping.

Array with conditions is a means of performing selective mapping based on the values of other AVPs, that is, not all AVPs are mapped, as DCA only considers or allows specific AVPs to be mapped if and only if a specified criterion is met.

Depending on the mapping type (inbound or outbound), the criteria involves matching the value of an AVP or an acsProfile against a list of known values.

Array with Conditions configuration

Here is the example Array with Conditions configuration in the Services `AVPMappings` section of the **eserv.config**.

```
{ # Array with Conditions Example
    AVPs = [ # Root-Level AVPs
        {
            AvpCodes = [ # 1st Entry in Root-Level AVPs
                {
                    avpCode = 7020
                }
            ]
            AVPs = [ # 1st-Level AVPs
                { # 1st Entry in 1st-Level AVPs
                    AvpCodes = [
                        {
                            avpCode = 1000
                            repeating = true
                        }
                    ]
                    AVPs = [ # 2nd-Level AVPs
                        { # 1st Entry in 2nd-Level AVPs
                            AvpCodes = [
                                {
                                    avpCode = 2000
                                    repeating = true
```

```
    }
]
TypeCriteria = [
    { # 1st Type Criterion
        includeIfMatches = [ 1, 11, 101 ]
        AvpCodes = [
            {
                avpCode = 3000
            }
        ]
        avpFormat = "Integer32"
        encodedExtension = {
            profileTag = 80100
            profileFormat = "INTEGER"
        }

        # AVPs below will be included if a match is found
    for 1st Type Criterion
        AVPs = [ # Conditional AVPs for 1st Type
    Criterion
            {
               AvpCodes = [
                   {
                       avpCode = 3001
                       repeating = true
                   }
               ]
               avpFormat = "UTF8String"
               encodedExtension = {
                   profileTag = 80101
                   profileFormat = "STRING"
               }# End of Conditional AVPs for 1st Type
    Criterion
            }
        ]
    } # End of 1st Type Criterion
    { # 2nd Type Criterion
       includeIfMatches = [ 2, 22, 202 ]
       AvpCodes = [
            {
                avpCode = 3000
            }
       ]
       avpFormat = "Integer32"
       encodedExtension = {
           profileTag = 80100
           profileFormat = "INTEGER"
       }

       # AVPs below will be included if a match is found
    for 2nd Type Criteria
       AVPs = [ # Conditional AVPs for 2nd Type Criterion
            {
              AvpCodes = [
                 {
                      avpCode = 3001
                      repeating = true
                 }
              ]
               avpFormat = "UTF8String"
               encodedExtension = {
                  profileTag = 80101
                  profileFormat = "STRING"
               }
```

```
                                    }
                                ] # End of Conditional AVPs for 2nd Type Criterion
                            } # End of 2nd Type Criterion
                        ] # End of TypeCriteria
                    } # End of 1st Entry of 2nd-Level AVPs
                ] # End of 2nd-Level AVPs
            } # End of 1st Entry of 1st-Level AVPs
        ] # End of 1st-Level AVPs
    } # End of 1st Entry of Root-Level AVPs
] # End of Root-Level AVPs

# Specify mapping applies INBOUND only.
mappingTypes = ["InitialRequest", "UpdateRequest", "EventRequest",
"TerminateRequest"]

avpFormat = "Grouped"
encodedExtension = {
    profileTag = 8020
    profileFormat = "ARRAY"
}
}
```

Array with Conditions - inbound - example 1

Here is a worked example of an array with conditions: inbound configuration.

Map only matching entry, not condition (AVP(3000)==1) inbound

```
{
    AVPs = [
        {
            AvpCodes = [
                { avpCode = 1000 }
            ]
            AVPs = [
                {
                    AvpCodes = [
                        { avpCode = 2000             # <-- 1000.2000
                          repeating = true           # <--      2000 is repeating
                        }
                    ]
                    TypeCriteria = [
                        {
                            includeIfMatches = [ 1 ]
                            AvpCodes = [
                                { avpCode = 3000     # <-- 1000.2000.3000 Inbound
                                Only
                                }
                            ]
                            avpFormat = "Integer32"   # <-- Inbound Only

                            # This is what is included if match found
                            AVPs = [
                                {
                                    AvpCodes = [
                                        { avpCode = 3001   # <-- 1000.2000.3001
                                          repeating = true # <--           3001 is
                                    repeating
                                        }
                                    ]
                                    avpFormat = "UTF8String"
                                    encodedExtension = {
```

```
                                        profileTag = 100    # Array element in sub-
                                    profile block
                                        profileFormat = "STRING"
                                    }
                                }
                            ]
                        }
                    ]
                }
            ]
        }
    ]
    # Specify mapping applies INBOUND only.
    mappingTypes = ["InitialRequest", "UpdateRequest", "EventRequest",
    "TerminateRequest"]

    encodedExtension = {
        profileTag = 8020
        profileFormat = "ARRAY"
    }
}
```

Mapping

This table shows the mapping of Diameter AVPs to ACS profile blocks resulting from the worked example array with conditions configuration section above. This example is for inbound. For outbound the arrows are reversed.

Diameter AVP				Profile Block			
Root AVP	1st Level	2nd Level	Value	Profile Block (ARRAY)	Sub-Profile Block	Tag	Value
1000	2000	3000	1	8000	1	100	"6449016000"
		3001	"6449016000"		2	100	"6449016001"
		3001	"6449016001"		3	100	"666666666"
	2000	3000	2				
		3001	"123123123"				
		3001	"6449016000"				
		3001	"6449016001"				
		3001	"6449016002"				
		3001	"6449016003"				
	2000	3000	1				
		3001	"666666666"				

Array with Conditions - Inbound - example 2

Here is an example of the array with conditions configuration in the Services `AVPMappings` section.

Map both matching entry and condition.

```
{
    AVPs = [
        {
            AvpCodes = [
                { avpCode = 1000 }
            ]
            AVPs = [
                {
                    AvpCodes = [
                        { avpCode = 2000              # <-- 1000.2000
                          repeating = true            # <--      2000 is repeating
                        }
                    ]
```

```
                    TypeCriteria = [
                        {
                            includeIfMatches = [ 1 ]
                            AvpCodes = [
                                { avpCode = 3000      # <-- 1000.2000.3000 Inbound
                                Only
                                }
                            ]
                            avpFormat = "Integer32"   # <-- Inbound Only

                            # This is what is included if match found
                            AVPs = [
                                {
                                    AvpCodes = [
                                        { avpCode = 3000   # <-- 1000.2000.3000
                                        }
                                    ]
                                    avpFormat = "Integer32"
                                    encodedExtension = {
                                        profileTag = 99   # Array element in sub-
                                profile block
                                        profileFormat = "INTEGER"
                                    }
                                }
                                {
                                    AvpCodes = [
                                        { avpCode = 3001   # <-- 1000.2000.3001
                                          repeating = true # <--           3001 is
                                repeating
                                        }
                                    ]
                                    avpFormat = "UTF8String"
                                    encodedExtension = {
                                        profileTag = 100   # Array element in sub-
                                profile block
                                        profileFormat = "STRING"
                                    }
                                }
                            ]
                        }
                    ]
                }
            ]
        }
    ]
    # Specify mapping applies INBOUND only.
    mappingTypes = ["InitialRequest", "UpdateRequest", "EventRequest",
    "TerminateRequest"]

    encodedExtension = {
        profileTag = 8020
        profileFormat = "ARRAY"
    }
}
```

Mapping

This table shows the mapping of Diameter AVPs to ACS Profile Blocks resulting from the worked example array with conditions configuration section above. This example is for inbound.

Diameter AVP				Profile Block			
Root AVP	1st Level	2nd Level	Value	Profile Block (ARRAY)	Sub-Profile Block	Tag	Value
1000	2000	3000	1	8000	1	99	1
		3001	"6449016000"			100	"6449016000"
		3001	"6449016001"		2	99	1
	2000	3000	2			100	"6449016001"
		3001	"123123123"		3	99	1
		3001	"6449016000"			100	"666666666"
		3001	"6449016001"				
		3001	"6449016002"				
		3001	"6449016003"				
	2000	3000	1				
		3001	"666666666"				

Array with Conditions - outbound - example 1

Here is the example array with conditions: outbound configuration.

Map only profileTag100. No typeCriteria against a profileTag. Outgoing AVP 3000 is set using a literal.

```
{
    AVPs = [
        {
            AvpCodes = [
                { avpCode = 1000 }
            ]
            AVPs = [
                {
                    AvpCodes = [
                        { avpCode = 2000              # <-- 1000.2000
                        }
                    ]

                    # Just map as follows (no type Criteria specified)
                    AVPs = [
                        {
                            AvpCodes = [
                                { avpCode = 3000   # <-- 1000.2000.3000
                                }
                            ]
                            avpFormat = "Integer32"
                            literal = 1            # i.e. Outbound only (*not*
                            mapping from profileTag99)
                        }
                        {
                            AvpCodes = [
                                { avpCode = 3001   # <-- 1000.2000.3001
                                  repeating = true # <--           3001 is repeating
                                }
                            ]
                            avpFormat = "UTF8String"
                            encodedExtension = {
                                profileTag = 100   # Array element in sub-profile
                                block
                                profileFormat = "STRING"
                            }
```

```
                        }
                    ]
                }
            ]
        }
    ]
    # Specify mapping applies OUTBOUND only.
    mappingTypes = ["InitialResponse", "UpdateResponse", "EventResponse",
    "TerminateResponse"]

    encodedExtension = {
        profileTag = 8000
        profileFormat = "ARRAY"
    }
}
```

Mapping

This table shows the mapping ACS profile blocks to AVPs in the example array with conditions configuration section in this topic. This example is for outbound.

Profile Block				Diameter AVP			
Profile Block (ARRAY)	Sub-Profile Block	Tag	Value	Root AVP	1st Level AVP	2nd Level AVP	Value
8000	1	99	1	1000	2000	3000	1
		100	"6449016000"			3001	"6449016000"
		101	"something"			3001	"6449016001"
	2	99	1			3001	"666666666"
		100	"6449016001"				
		101	"something"				
		102	"else"				
	3	99	1				
		100	"666666666"				
		101	"other"				

Array with Conditions - outbound - example 2

Here is the example array with conditions: outbound configuration.

Map only profileTag 100. No typeCriteria against a profileTag. Outgoing AVP 3000 is set using a literal.

```
{
    AVPs = [
        {
            AvpCodes = [
                { avpCode = 1000 }
            ]
            AVPs = [
                {
                    AvpCodes = [
                        { avpCode = 2000              # <-- 1000.2000
                          repeating = true            # <--      2000 is repeating
                        }
                    ]

                    # Just map as follows (no type Criteria specified)
                    AVPs = [
                        {
                            AvpCodes = [
                                { avpCode = 3000    # <-- 1000.2000.3000
                                }
```

```
                            ]
                            avpFormat = "Integer32"
                            literal = 1             # i.e. Outbound only (*not*
                            mapping from profileTag99)
                        }
                        {
                            AvpCodes = [
                                { avpCode = 3001    # <-- 1000.2000.3001
                                }
                            ]
                            avpFormat = "UTF8String"
                            encodedExtension = {
                                profileTag = 100    # Array element in sub-profile
                                block
                                profileFormat = "STRING"
                            }
                        }
                    ]
                }
            ]
        }
    ]
    # Specify mapping applies OUTBOUND only.
    mappingTypes = ["InitialResponse", "UpdateResponse", "EventResponse",
    "TerminateResponse"]

    encodedExtension = {
        profileTag = 8000
        profileFormat = "ARRAY"
    }
}
```

Mapping

This table shows the mapping ACS profile blocks to AVPs in the example array with conditions configuration section in this topic. This example is for outbound.

Profile Block				Diameter AVP			
Profile Block (ARRAY)	**Sub-Profile Block**	**Tag**	**Value**	**Root AVP**	**1st Level AVP**	**2nd Level AVP**	**Value**
8000	1	100	"6449016000"	1000	2000	3000	1
		101	"something"			3001	"6449016000"
		102	"else"		2000	3000	1
		103	"and more"			3001	"6449016001"
	2	100	"6449016001"		2000	3000	1
		101	"another"			3001	"666666666"
		102	"something"				
		103	"more"				
	3	100	"666666666"				
		100	"and"				
		101	"others"				

Array with Conditions - outbound - example 3

Here is the example array with conditions: outbound configuration.

Map only profileTag 100 with typeCriteria specified against profileTag 99. Outgoing AVP 3000 is set based on profileTag 99.

```
{
    AVPs = [
        {
```

```
            AvpCodes = [
                { avpCode = 1000 }
            ]
            AVPs = [
                {
                    AvpCodes = [
                        {avpCode = 2000           # <-- 1000.2000
                        }
                    ]
                    TypeCriteria = [
                        {
                            includeIfMatches = [ 1 ]  # <-- Only profileTag(99)==1
                            are mapped
                            encodedExtension = {
                                profileTag = 99   # Array element in sub-profile
                                block
                                profileFormat = "INTEGER"
                            }

                            # This is what is included if match found:
                            AVPs = [
                                {
                                    AvpCodes = [
                                        { avpCode = 3000   # <-- 1000.2000.3000
                                        }
                                    ]
                                    avpFormat = "Integer32"
                                    encodedExtension = {
                                        profileTag = 99   # Also include the matched
                                value(1) in outbound msg
                                        profileFormat = "INTEGER"
                                    }
                                }
                                {
                                    AvpCodes = [
                                        { avpCode = 3001   # <-- 1000.2000.3001
                                          repeating = true # <--           3001 is
                                repeating
                                        }
                                    ]
                                    avpFormat = "UTF8String"
                                    encodedExtension = {
                                        profileTag = 100   # Array element in sub-
                                profile block
                                        profileFormat = "STRING"
                                    }
                                }
                            ]
                        }
                    ]
                }
            ]
        }
    ]
    # Specify mapping applies OUTBOUND only.
    mappingTypes = ["InitialResponse", "UpdateResponse", "EventResponse",
    "TerminateResponse"]

    encodedExtension = {
        profileTag = 8000
        profileFormat = "ARRAY"
    }
}
```

Mapping

This table shows the mapping ACS profile blocks to AVPs in the example array with conditions configuration section in this topic. This example is for outbound.

Profile Block				Diameter AVP			
Profile Block (ARRAY)	Sub-Profile Block	Tag	Value	Root AVP	1st Level AVP	2nd Level AVP	Value
8000	1	99	1	1000	2000	3000	1
		100	"6449016000"			3001	"6449016000"
		101	"something"			3001	"666666666"
	2	99	2				
		100	"6449016001"				
		101	"something"				
		102	"else"				
	3	99	1				
		100	"666666666"				
		101	"others"				

Array with Conditions - outbound - example 4

Here is the example array with conditions: outbound configuration.

Map only profileTag 100 with typeCriteria specified against profileTag 99. No AVP 3000 in outgoing diameter message.

```
{
    AVPs = [
        {
            AvpCodes = [
                { avpCode = 1000 }
            ]
            AVPs = [
                {
                    AvpCodes = [
                        {avpCode = 2000           # <-- 1000.2000
                        }
                    ]
                    TypeCriteria = [
                        {
                            includeIfMatches = [ 1 ]  # <-- Only profileTag(99)==1
                            are mapped
                            encodedExtension = {
                                profileTag = 99   # Array element in sub-profile
                                block
                                profileFormat = "INTEGER"
                            }

                            # This is what is included if match found:
                            AVPs = [
                                    AvpCodes = [
                                        { avpCode = 3001   # <-- 1000.2000.3001
                                          repeating = true # <--           3001 is
                                repeating
                                    }
                                ]
                                avpFormat = "UTF8String"
                                encodedExtension = {
                                    profileTag = 100   # Array element in sub-
                                profile block
                                    profileFormat = "STRING"
                                }
```

```
                                    }
                                ]
                            }
                        ]
                    }
                ]
            }
        ]
        # Specify mapping applies OUTBOUND only.
        mappingTypes = ["InitialResponse", "UpdateResponse", "EventResponse",
        "TerminateResponse"]

        encodedExtension = {
            profileTag = 8000
            profileFormat = "ARRAY"
        }
}
```

Mapping

This table shows the mapping ACS profile blocks to AVPs in the example array with conditions configuration section in this topic. This example is for outbound.

Profile Block					Diameter AVP			
Profile Block (ARRAY)	Sub-Profile Block	Tag	Value		Root AVP	1st Level AVP	2nd Level AVP	Value
8000	1	99	1		1000	2000	3001	"6449016000"
		100	"6449016000"				3001	"666666666"
		101	"something"					
	2	99	2					
		100	"6449016001"					
		101	"something"					
		102	"else"					
	3	99	1					
		100	"666666666"					
		101	"others"					

Array with Context

Introduction

A key array has a limitation in that it cannot handle the situation when the possible key values are not well known, for example, in cases when we may not have a unique key, or the key might otherwise rely on multiple items from the hierarchy. For these cases, you may use Array with Context mapping.

The key here is that DCA allows other sub-AVPs in a hierarchy to be marked as a Context AVP, using the parameter setting `contextAVP = true`. All AVPs marked as a Context AVP then collectively make the items which provide context.

Array with Context configuration

Here is the example Array with Context configuration in the Services `AVPMappings` section of the **eserv.config**.

```
{ # Array with Context Example
    AVPs = [ # Root-Level AVPs
        { # 1st Entry of Root-Level AVPs
            AvpCodes = [
```

```
        {
            avpCode = 7030
        }
    ]
    AVPs = [ # 1st-Level AVPs
        { # 1st Entry of 1st-Level AVPs
            AvpCodes = [
                {
                    avpCode = 1000
                }
            ]
            AVPs = [ # 2nd-Level AVPs
                { # 1st Entry of 2nd-Level AVPs
                    AvpCodes = [
                        {
                            avpCode = 2000
                            repeating = true
                        }
                    ]
                    AVPs = [ # 3rd-Level AVPs
                        { # Context AVP
                            AvpCodes = [
                                {
                                     avpCode = 3000   # This is AVP
                        7030.1000.2000.3000
                                }
                            ]
                            contextAVP = true
                            avpFormat = "UTF8String"
                            encodedExtension = {
                                profileTag = 80301
                                profileFormat = "STRING"
                            }
                        }
                        { # Data AVP
                            AvpCodes = [
                                {
                                    avpCode = 3001   # This is AVP
                        7030.1000.2000.3001
                                    repeating = true
                                }
                            ]
                            avpFormat = "UTF8String"
                            encodedExtension = {
                                profileTag = 80303
                                profileFormat = "STRING"
                            }
                        } # End of Data AVP
                    ] # End of 3rd-Level AVPs
                } # End of 1st Entry of 2nd-Level AVPs
                { # 2nd Entry of 2nd-Level AVPs(Context AVP)
                    AvpCodes = [
                        {
                            avpCode = 2001    # This is AVP 7030.1000.2001
                        }
                    ]
                    contextAVP = true
                    avpFormat = "UTF8String"
                    encodedExtension = {
                        profileTag = 80302
                        profileFormat = "STRING"
                    }
                } # End of 2nd Entry of 2nd-Level AVPs (Content AVP)
            ] # End of 2nd-Level AVPs
```

```
            } # End of 1st Entry of 1st-Level AVPs
        ] # End 1st-Level AVPs
    } # End of 1st Entry of Root-Level AVPs
] # End of Root-Level AVPs

# Specify mapping applies INBOUND only.
mappingTypes = ["InitialRequest", "UpdateRequest", "EventRequest",
"TerminateRequest"]

avpFormat = "Grouped"
encodedExtension = {
    profileTag = 8030
    profileFormat = "ARRAY"
}
```

} # End of Array with Context Example

Array with Context - inbound example

Here is an example of the Array with Context configuration in the Services `AVPMappings` section.

```
{
    AVPs = [
        {
            AvpCodes = [
                {
                    avpCode = 1000
                }
            ]
            AVPs = [
                {
                    AvpCodes = [
                        {
                            avpCode = 2000     # <-- 1000.2000
                            repeating = true   # <--      2000 is repeating
                        }
                    ]
                    AVPs = [
                        {
                            AvpCodes = [
                                { avpCode = 3000  # <-- 1000.2000.3000
                                }
                            ]
                            avpFormat = "UTF8String"
                            encodedExtension = {
                                profileTag = 101   # Array element in sub-profile
                                block
                                profileFormat = "STRING"
                            }
                            contextAVP = true
                        }
                        {
                            AvpCodes = [
                                { avpCode = 3001   # <-- 1000.2000.3001
                                  repeating = true # <--           3001 is repeating
                                }
                            ]
                            avpFormat = "UTF8String"
                            encodedExtension = {
                                profileTag = 102   # <-- Array element in sub-profile
                                block
                                profileFormat = "STRING"
                            }
```

```
                        }
                    ]
                }
                {
                    AvpCodes = [
                        { avpCode = 2001    # <-- 1000.2001
                        }
                    ]
                    avpFormat = "UTF8String"
                    encodedExtension = {
                        profileTag = 100   # <-- Array element in sub-profile block
                        profileFormat = "STRING"
                    }
                    contextAVP = true
                }
            ]
        }
    ]

    # Specify mapping applies INBOUND only.
    mappingTypes = ["InitialRequest", "UpdateRequest", "EventRequest",
    "TerminateRequest"]

    encodedExtension = {
        profileTag = 8000
        profileFormat = "ARRAY"
    }
}
```

Mapping

This table shows the mapping AVPs to ACS profile blocks in the example array with context configuration section in this topic. This example is for inbound.

Diameter Message				Profile Block			
Root AVP	1st Level	2nd Level	Value	Profile Block (ARRAY)	Sub-Profile Block	Tag	Value
1000	2000	3000	"ContextB"	8000	1	100	"RootContext"
		3001	"6449016000"			101	"ContextB"
		3001	"6449016001"			102	"6449016000"
	2001		"RootContext"		2	100	"RootContext"
	2000	3000	"ContextC"			101	"ContextB"
		3001	"123123123"			102	"6449016001"
					3	100	"RootContext"
						101	"ContextC"
						102	"123123123"

Array with Context - outbound example 1

Here is an example of the array with context outbound configuration in the Services `AVPMappings` section.

AVP 3001 is not repeating in the outbound Diameter message.

```
{
    AVPs = [
        {
            AvpCodes = [
                {
                    avpCode = 1000
                }
            ]
```

```
            AVPs = [
                {
                    AvpCodes = [
                        {
                            avpCode = 2000     # <-- 1000.2000
                            repeating = true   # <--      2000 is repeating
                        }
                    ]
                    AVPs = [
                        {
                            AvpCodes = [
                                { avpCode = 3000  # <-- 1000.2000.3000
                                }
                            ]
                            avpFormat = "UTF8String"
                            encodedExtension = {
                                profileTag = 101   # Array element in sub-profile
                                block
                                profileFormat = "STRING"
                            }
                            contextAVP = true
                        }
                        {
                            AvpCodes = [
                                { avpCode = 3001   # <-- 1000.2000.3001 (not
                                repeating)
                                }
                            ]
                            avpFormat = "UTF8String"
                            encodedExtension = {
                                profileTag = 102   # <-- Array element in sub-profile
                                block
                                profileFormat = "STRING"
                            }
                        }
                    ]
                }
                {
                    AvpCodes = [
                        { avpCode = 2001    # <-- 1000.2001
                        }
                    ]
                    avpFormat = "UTF8String"
                    encodedExtension = {
                        profileTag = 100   # <-- Array element in sub-profile block
                        profileFormat = "STRING"
                    }
                    contextAVP = true
                }
            ]
        }
    ]

    # Specify mapping applies OUTBOUND only.
    mappingTypes = ["InitialResponse", "UpdateResponse", "EventResponse",
    "TerminateResponse"]

    encodedExtension = {
        profileTag = 8000
        profileFormat = "ARRAY"
    }
}
```

Mapping

AVP 3001 is not repeating in the outbound Diameter message.

Profile Block				Diameter AVP			
Profile Block (ARRAY)	Sub-Profile Block	Tag	Value	Root AVP	1st Level AVP	2nd Level AVP	Value
8000	1	100	"RootContext"	1000	2000	3000	"ContextB"
		101	"ContextB"			3001	"6449016000"
		102	"6449016000"		2000	3000	"ContextB"
	2	100	"RootContext"			3001	"6449016001"
		101	"ContextB"		2000	3000	"ContextC"
		102	"6449016001"			3001	"123123123"
	3	100	"RootContext"		2001		"RootContext"
		101	"ContextC"				
		102	"123123123"				

Array with Context - outbound example 2

Adding `repeating` to the AVP 3001 configuration shown in *Array with Context - outbound example 1* (on page 106), produces the mapping shown in mapping example 2.

```
AvpCodes = [
    { avpCode = 3001   # <-- 1000.2000.3001
      repeating = true # <--           3001 is repeating
}
```

Mapping

AVP 3001 is repeating in the outbound Diameter message.

Profile Block				Diameter AVP			
Profile Block (ARRAY)	Sub-Profile Block	Tag	Value	Root AVP	1st Level AVP	2nd Level AVP	Value
8000	1	100	"RootContext"	1000	2000	3000	"ContextB"
		101	"ContextB"			3001	"6449016000"
		102	"6449016000"			3001	"6449016001"
	2	100	"RootContext"		2000	3000	"ContextC"
		101	"ContextB"			3001	"123123123"
		102	"6449016001"		2001		"RootContext"
	3	100	"RootContext"				
		101	"ContextC"				
		102	"123123123"				

Conditional AVP

Introduction

Conditional AVP enables you to perform a mapping based on the value of another AVP. For example, we might want to map the Service-Parameter-Value AVP in a grouped Service-Parameter-Info AVP to a profile field, but only if its type (specified in the Service-Parameter-Type AVP) is one we are interested in. The conditional AVP includes the `typeCriteria` array to specify the condition to match.

For outbound mapping, conditional AVPs enable mapping to be performed based on the value:

- In a profile block, and/or
- Of another AVP in the outbound message being constructed

Conditional AVP configuration

Here is an example of the Conditional AVP configuration in the Services `AVPMappings` section of the **eserv.config**.

```
{
    AVPs = [ # Root-Level AVPs
        { # 1st Entry of Root-Level AVPs
            AvpCodes = [
                {
                    avpCode = 4000
                }
            ]
            TypeCriteria = [
                {  # 1st Criterion
                    AvpCodes = [
                        {
                            avpCode = 4001
                        }
                    ]
                    includeIfMatches = [ 1, 10, 101, 1001, 10001 ]
                    avpFormat = "Integer32"

                    # This is the AVP that is searched and map if a match is found
                    AVPs = [ # Conditional AVPs for 1st Criterion
                        {
                            AvpCodes = [
                                {
                                    avpCode = 4002
                                }
                            ]
                            avpFormat = "UTF8String"
                            encodedExtension = {
                                profileTag = 99123
                                profileFormat = "STRING"
                            }
                        }
                    ] # End of Conditional AVPs for 1st Criterion
                } # End of 1st Criterion
                { # 2nd Criterion
                    AvpCodes = [
                        {
                            avpCode = 4001
                        }
                    ]
                    includeIfMatches = [ 2, 20, 202, 2002, 20002 ]
                    avpFormat = "Integer32"

                    # This is the AVP that is searched and map if a match is found
                    AVPs = [ # Conditional AVPs for 2nd Criterion
                        {
                            AvpCodes = [
                                {
                                    avpCode = 4002
                                }
                            ]
                            avpFormat = "UTF8String"
                            encodedExtension = {
                                profileTag = 99124
                                profileFormat = "STRING"
                            }
                        }
                    ] # End of Conditional AVPs for 2nd Criterion
```

```
                } # End of 2nd Criterion
            ] # End of TypeCriteria
        } # End of 1st Entry of Root-Level AVPs
    ] # End of Root-Level AVPs

    # Specify mapping applies INBOUND only.
    mappingTypes = ["InitialRequest", "UpdateRequest", "EventRequest",
    "TerminateRequest"]

    # Profile tag 99123 or 99124 is encoded as a child element inside an acsProfile
    Array.
    encodedExtension = {
        profileTag = 8000
        profileFormat = "ARRAY"
    }
} # End of Conditional AVP Example
```

Note: "Conditional AVPs" look similar to a non-repeating / non-ProfileArray case of "Array with Conditions". See *Array with Conditions - inbound - example 1* (on page 95).

If the `avpCode` being matched is not nested, you can map it without using `typeCriteria`, by configuring the condition outside `AVPs = [ ... ]`. See *Simple conditional* (on page 86).

Conditional AVP - inbound example 1

Here is an example of the conditional AVP configuration in the Services `AVPMappings` section.

Service-Parameter-Info (440)

- Service-Parameter-Type (441)
- Service-Parameter-Value (442)

Map only matching entry, not condition.

In this example, no parent level (ARRAY-type) encoding format is specified here as encoding specified in leaf or child element of AVPs above are encoded at the root level of the ACS profile block.

```
{
    AVPs = [
        {
            AvpCodes = [
                {
                    avpCode = 440
                }
            ]
            TypeCriteria = [
                {  # Criteria for match value [ 1 ]
                    includeIfMatches = [ 1 ]
                    AvpCodes = [
                        { avpCode = 441          # <--  Inbound only: 441
                        }
                    ]
                    # or if specifying a fully qualified path:
                    # AvpCodes = [
                    #  { avpCode = 440 }
                    #  { avpCode = 441 }
                    # ]
                    avpFormat = "Integer32"      # <--  Inbound only


                    # This is what to include if match found:
                    AVPs = [
                        {
                            AvpCodes = [
                                { avpCode = 442  # <-- 440.442
```

```
                                }
                            ]
                            avpFormat = "UTF8String"
                            encodedExtension = {
                                profileTag = 99123 # Array element in sub-profile
                                block
                                profileFormat = "STRING"
                            }
                        }
                    ]
                }
                {   # Criteria for match value [ 2 ]
                    includeIfMatches = [ 2 ]
                    AvpCodes = [
                        { avpCode = 441          # <--  Inbound only: 441
                        }
                    ]
                    avpFormat = "Integer32"      # <--  Inbound only

                     # This is what to include if match found:
                    AVPs = [
                        {
                            AvpCodes = [
                                { avpCode = 442  # <-- 440.442
                                }
                            ]
                            avpFormat = "UTF8String"
                            encodedExtension = {
                                profileTag = 99124 # Array element in sub-profile
                                block
                                profileFormat = "STRING"
                            }
                        }
                    ]
                }
            ]
        }
    ]

    # Specify mapping applies INBOUND only.
    mappingTypes = ["InitialRequest", "UpdateRequest", "EventRequest",
    "TerminateRequest"]

}
```

Mapping example 1

This table shows the mapping AVPs to ACS profile blocks in the example conditional AVPs configuration section in this topic. This example is for inbound.

Diameter AVP				Profile Block	
Root AVP	**1st Level**	**Value**		**Tag**	**Value**
440	441	1		99123	"Value for Type 1"
	442	"Value for Type 1"			

or

Diameter AVP				Profile Block	
Root AVP	**1st Level**	**Value**		**Tag**	**Value**
440	441	2		99124	"Value for Type 2"
	442	"Value for Type 2"			

Conditional AVP example 1

Here is an example of the conditional AVP configuration in the Services `AVPMappings` section.

Adding the following configuration to the AVPs to include if match is found to each criterion for match value produces the mapping shown in mapping example 2.

Map only matching entry, not condition.

```
{
    AvpCodes = [
        { avpCode = 441  # <-- 440.441
        }
    ]
    avpFormat = "Integer32"
    encodedExtension = {
        profileTag = 99001 # Array element in sub-profile block
        profileFormat = "INTEGER"
    }
}
```

Mapping example 2

This table shows the mapping AVPs to ACS profile blocks in the example conditional AVPs configuration section in this topic. This example is for inbound.

Diameter AVP				Profile Block	
Root AVP	**1st Level**	**Value**		**Tag**	**Value**
440	441	1		90001	1
	442	"Value for Type 1"		99123	"Value for Type 1"

or

Diameter AVP				Profile Block	
Root AVP	**1st Level**	**Value**		**Tag**	**Value**
440	441	2		90001	2
	442	"Value for Type 2"		99124	"Value for Type 2"

Conditional AVP example 3

Here is an example of the conditional AVP configuration in the Services `AVPMappings` section.

This is the same as *Conditional AVP - inbound example 1* (on page 110) but maps to an acsProfile ARRAY, by adding the following configuration to the end.

Map only matching entry, not condition, but map to an acsProfile ARRAY.

```
encodedExtension = {
    profileTag = 8000
    profileFormat = "ARRAY"
}
```

Mapping example 3

This table shows the mapping AVPs to ACS profile blocks in the example conditional AVPs configuration section in this topic. This example is for inbound.

Note similarity with the non-repeating array with conditions.

Diameter AVP				Profile Block			
Root AVP	1st Level	Value		Profile Block (ARRAY)	Sub-Profile Block	Tag	Value
440	441	1		80001	1	99123	"Value for Type 1"
	442	"Value for Type 1"					

Diameter AVP				Profile Block			
Root AVP	1st Level	Value		Profile Block (ARRAY)	Sub-Profile Block	Tag	Value
440	441	2		80001	2	99123	"Value for Type 2"
	442	"Value for Type 2"					

Conditional AVP - outbound example 1

Here is an example outbound conditional AVP configuration in the Services `AVPMappings` section.

No ACS profileTag conditions applicable when mapping to outbound Diameter message.

In this example, no parent level (ARRAY-type) encoding format is specified here as encoding specified in leaf or child element of AVPs above are encoded at the root level of the ACS Profile Block.

```
{
    AVPs = [
        {
            AvpCodes = [
                {
                    avpCode = 440
                }
            ]
            TypeCriteria = [

                {     # Criteria for match tag 99123
                    includeIfMatches = [ ]  # <-- Match any value as long as
                    profileTag 99123 is present
                    encodedExtension = {
                        profileTag = 99123          # <-- Outbound Only
                        profileFormat = "STRING"    # <-- Outbound Only
                    }

                    # This is what to include if match found:
                    AVPs = [
                        #Include type sub-AVP 1
                        {
                            AvpCodes = [
                                { avpCode = 441  # <-- 440.441
                                }
                            ]
                            avpFormat = "UTF8String"
                            encodedExtension = {
                                profileTag = 99123 # Array element in sub-profile
                                block
                                profileFormat = "STRING"
                            }
                        }
                    ]
                }
                {  # Criteria for match tag 99124
                    includeIfMatches = [ ]  # <-- Match any value as long as
                    profileTag 99124 is present
```

```
                    encodedExtension = {
                        profileTag = 99124          # <-- Outbound Only
                        profileFormat = "STRING"    # <-- Outbound Only
                    }

                     # This is what to include if match found:
                    AVPs = [
                        #Include type sub-AVP 2
                        {
                            AvpCodes = [
                                { avpCode = 441  # <-- 440.441
                                }
                            ]
                            avpFormat = "Integer32"
                            literal = 2      # not mapped from a tag in the profile
                            block
                            }
                        }
                        {
                            AvpCodes = [
                                { avpCode = 442  # <-- 440.442
                                }
                            ]
                            avpFormat = "UTF8String"
                            encodedExtension = {
                                profileTag = 99124 # Array element in sub-profile
                                block
                                profileFormat = "STRING"
                            }
                        }
                    ]
                }
            ]
        }
    ]

    # Specify mapping applies OUTBOUND only.
    mappingTypes = ["InitialResponse", "UpdateResponse", "EventResponse",
    "TerminateResponse"]

}
```

Mapping example 1 - conditional - outbound

This table shows the mapping ACS profile blocks to AVPs to in the example conditional AVPs configuration section in this topic. This example is for outbound.

Profile Block			Diameter AVP		
Tag	**Value**		**Root AVP**	**1st Level**	**2nd Level**
99123	"Value for Type 1"		440	441	1
				442	"Value for Type 1"

or

Profile Block			Diameter AVP		
Tag	**Value**		**Root AVP**	**1st Level**	**2nd Level**
99124	"Value for Type 2"		440	441	2
				442	"Value for Type 2"

Conditional AVP - outbound example 2

Here is an example outbound conditional AVP configuration in the Services `AVPMappings` section.

Only the matching entries in ACS Profile Block are mapped in outbound Diameter message (Condition is "profileTag(90001) == 2")

In this example, no parent level (ARRAY-type) encoding format is specified here as encoding specified in leaf or child element of AVPs above are encoded at the root level of the ACS Profile Block.

```
{
    AVPs = [
        {
            AvpCodes = [
                {
                    avpCode = 440
                }
            ]
            TypeCriteria = [

                {   # Criteria for match value [1] In this example, this is NOT the
                matching entry.
                    includeIfMatches = [ 1 ]
                    encodedExtension = {
                        profileTag = 90001          # <-- Outbound Only
                        profileFormat = "STRING"    # <-- Outbound Only
                    }

                    # This is what to include if match found:
                    AVPs = [
                        #Include type sub-AVP (the matching condition)
                        {
                            AvpCodes = [
                                { avpCode = 441  # <-- 440.441
                                }
                            ]
                            avpFormat = "Integer32"
                            encodedExtension = {
                                profileTag = 90001 # Array element in sub-profile
                                block
                                profileFormat = "INTEGER"
                            }
                        }
                        {
                            AvpCodes = [
                                { avpCode = 442  # <-- 440.442
                                }
                            ]
                            avpFormat = "UTF8String"
                            encodedExtension = {
                                profileTag = 99123 # Array element in sub-profile
                                block
                                profileFormat = "STRING"
                            }
                        }
                    ]
                }
                {  # Criteria for match value [2] In this example, this is the
                matching entry.
                    includeIfMatches = [ 2 ]
                    encodedExtension = {
                        profileTag = 90001          # <-- Outbound Only
                        profileFormat = "INTEGER"    # <-- Outbound Only
                    }

                     # This is what to include if match found:
                    AVPs = [
```

```
                        #Include type sub-AVP (the matching condition)
                        {
                            AvpCodes = [
                                { avpCode = 441  # <-- 440.441
                                }
                            ]
                            avpFormat = "Integer32"
                            encodedExtension = {
                                profileTag = 90001 # Array element in sub-profile
                                block
                                profileFormat = "INTEGER"
                            }
                        }
                        {
                            AvpCodes = [
                                { avpCode = 442  # <-- 440.442
                                }
                            ]
                            avpFormat = "UTF8String"
                            encodedExtension = {
                                profileTag = 99124 # Array element in sub-profile
                                block
                                profileFormat = "STRING"
                            }
                        }
                    ]
                }
            ]
        }
    ]

    # Specify mapping applies OUTBOUND only.
    mappingTypes = ["InitialResponse", "UpdateResponse", "EventResponse",
    "TerminateResponse"]

}
```

Mapping example 2 - conditional - outbound

This table shows the mapping ACS profile blocks to AVPs to in the example conditional AVPs configuration section in this topic. This example is for outbound.

Profile Block			Diameter AVP		
Tag	**Value**		**Root AVP**	**1st Level**	**2nd Level**
90001	2		440	441	2
99124	"Value for Type 2"			442	"Value for Type 2"

Prefix Tree

Introduction

Prefix Tree enables you to map repeating AVPs to and from a prefix tree ACS profile block, specified using `profileFormat = "PREFIXTREE"`.

Prefix Tree configuration

Here is an example of the Prefix Tree configuration in the Services `AVPMappings` section of the **eserv.config** file.

```
{
    AVPs = [ # Root-Level AVPs
```

```
        {
            AvpCodes = [
                {
                    avpCode = 7100
                    vendorId = "<Vendor ID>"  # Optional. Vendor specific AVP, if
                    present.
                }
            ]
            AVPs = [ # 1st-Level AVPs
                {
                    AvpCodes = [
                        {
                            avpCode = 1000
                        }
                    ]
                    AVPs =  [ # 2nd-Level AVPs
                        {
                            AvpCodes = [
                                {
                                    avpCode = 2000
                                    repeating = true
                                }
                            ]
                            avpFormat = "UTF8String"
                        }
                    ] # End of 2nd-Level AVPs
                }
            ] # End of 1st-Level AVPs
        }
    ] # End of Root-Level AVPs
    avpFormat = "Grouped"
    encodedExtension = {
        profileTag = 8100
        profileFormat = "PREFIXTREE"
    }

    # Specify mapping applies INBOUND only.
    mappingTypes = ["InitialRequest", "UpdateRequest", "EventRequest",
    "TerminateRequest"]

    # AVPs above are encoded into a PREFIXTREE-type in an ACS Profile Block.
    encodedExtension = {
        profileTag = 7000
        profileFormat = "PREFIXTREE"
    }
    } # End of Prefix Tree example
```

Prefix Tree example

Here is an example of the prefix tree configuration in the Services `AVPMappings` section.

Note that the parent tag is specified outside of the AVPs array block. Parent tag 7000 has type "PREFIXTREE", hence `profileTag` and `profileFormat` for child elements are not applicable as this child AVP is encoded as entries within a prefix tree.

```
{
    AvpCodes = [
        avpCode = 1000
    ]
    AVPs = [
        {
            AvpCodes = [
```

```
                {
                    avpCode = 2000              # <-- 1000.2000
                    repeating = true            # <--      2000 is repeating
                }
            ]
            avpFormat = "UTF8String"

        }
    ]
    # Specify mapping applies INBOUND only.
    mappingTypes = ["InitialRequest", "UpdateRequest", "EventRequest",
    "TerminateRequest"]

    encodedExtension = {
        profileTag = 7000
        profileFormat = "PREFIXTREE"
    }
}
```

To specify outbound, the `mappingTypes` are specified as:

```
    # Specify mapping applies OUTBOUND only.
    mappingTypes = ["InitialResponse", "UpdateResponse", "EventResponse",
    "TerminateResponse"]
```

Mapping

This table shows the mapping AVPs to ACS profile blocks in the example prefix tree configuration section in this topic. This example is for inbound. For outbound, the mapping direction is reversed.

Diameter AVP				Profile Block	
Root AVP	**1st Level**	**Value**		**Profile Block (Prefix Tree)**	**Value**
1000	2000	"049772"		7000	"049772"
	2000	"644901"			"644901"
	2000	"0800500"			"0800500"

Timestamp

Introduction

The DCA interface on the SLC is able to record the time that the:

- Credit-Control-Request (CCR) was received (Time-In)
- Credit-Control-Answer (CCA) was ready to be assembled and sent (Time-Out)

This allows you to track processing time and, for example, identify bottle-necks.

You can map:

- The time a CCR was received into a configurable AVP in the CCA message
- The time a Credit-Control processing for a request was completed, into a configurable AVP in the CCA message

You can copy the timestamp from the incoming Diameter message to the outgoing Diameter message.

Timestamp example 1

This example copies a timestamp (the 3GPP eventtimestamp) from the incoming Diameter message to the outgoing Diameter message.

The data is copied through the profile tag 6291458; this is special cased to be copied from the incoming profile block to the outgoing one. It is an ARRAY tag, so that you can put whatever data you like in it.

```
{
    mappingTypes = ["InitialRequest", "InitialResponse", "UpdateRequest",
    "UpdateResponse"]
    avpFormat = "Grouped"
    extensionFormat = "encoded"
    encodedExtension = {
        profileTag = 6291458
        profileFormat = "ARRAY"
    }
    AVPs = [
        {
            AvpCodes = [
                {
                    avpCode = 833
                    vendorId = 10415
                }
                {
                    avpCode = 6
                    vendorId = 16247
                    repeating = True
                }
            ]
            avpFormat = "Integer32"
            extensionFormat = "encoded"
            encodedExtension = {
                profileTag = 8192004
                profileFormat = "INTEGER"
            }
        }
    ]
}
```

Timestamp example 2

This example places a received timestamp in the outgoing message. This uses the same repeating AVP as the previous example, so it gets appended to the group.

```
{
    mappingTypes = [ "InitialResponse", "UpdateResponse" ]
    # Any Integer/Unsigned 32/64 or OctetString may be used.
    avpFormat = "Integer32"
    timestamp = "TIME_REQUEST_RECEIVED32"
    AvpCodes = [
        {
            avpCode = 833
            vendorId = 10415
        }
        {
            avpCode = 6
            vendorId = 16247
            repeating = True
        }
    ]
}
```

Note: See the *timestamp* (on page 76) parameter description for a list of values.

Timestamp example 3

This example places a replying timestamp in the outgoing message.

```
{
    mappingTypes = [ "InitialResponse", "UpdateResponse" ]
```

```
    avpFormat = "Integer64"
    timestamp = "TIME_NOW64"
    AvpCodes = [
        {
            avpCode = 833
            vendorId = 10415
        }
        {
            avpCode = 7
            vendorId = 16247
            repeating = True
        }
    ]
}
```

RAR Example

```
AvpMappings = [
            #
            # Request Mapping set
            #
            {
                  AVPs = [
                      {
                          AvpCodes = [
                              {
                                  # Session-Id
                                  avpCode = 263
                                  vendorId = -1
                              }
                          ]
                          avpFormat = "UTF8String"
                          extensionFormat = "encoded"
                          encodedExtension = {
                              profileTag = 6291461
                              profileFormat = "STRING"
                          }
                      }
                      {
                          AvpCodes = [
                              {
                                  # Origin-Host
                                  avpCode = 264
                                  vendorId = -1
                              }
                          ]
                          avpFormat = "UTF8String"
                          extensionFormat = "encoded"
                          encodedExtension = {
                              profileTag = 6291466
                              profileFormat = "STRING"
                          }
                      }
                  ]
            }


            #
            # Response Mapping set
            #
            {
             mappingTypes = [
               "InitialResponse"
               "UpdateResponse"
```

```
            ]
              AVPs = [
              {
                  AvpCodes = [
                  {
                        # da_final_unit_indication
                        avpCode = 430
                        vendorId = -1
                  }
                  ]
                    AVPs = [
                    {
                      AvpCodes = [
                      {
                          # da_redirect_server
                          avpCode = 434
                          vendorId = -1
                      }
                      ]
                        AVPs = [
                        {
                          AvpCodes = [
                          {
                              # da_redirect_address_type
                              avpCode = 433
                              vendorId = -1
                          }
                          ]
                          #avpFormat = "Enumerated"
                          avpFormat = "UTF8String"
                          extensionFormat = "encoded"
                          encodedExtension = {
                              # DCA Redirect Address Type
                              profileTag = 6291464
                              profileFormat = "STRING"
                          }
                        }
                        {
                          AvpCodes = [
                          {
                              # da_redirect_address
                              avpCode = 435
                              vendorId = -1
                          }
                          ]
                          avpFormat = "UTF8String"
                          extensionFormat = "encoded"
                          encodedExtension = {
                              # DCA Redirect Address
                              profileTag = 6291465
                              profileFormat = "STRING"
                          }
                        }
                        ] # 3rd level
                    }
                    ] # 2nd level
              }
              ] # 1st level
          }

] # end of AVP Mappings
```

Chapter 5

Control Plans

Overview

Introduction

This chapter explains the example Control Plans that are shipped with Oracle Communications Network Charging and Control (NCC) Diameter Control Agent.

These are sufficient to run simple Diameter services. There are:

- for event based services:
  - CHECK_BALANCE
  - DIRECT_DEBITING
  - PRICE_ENQUIRY
  - REFUND_ACCOUNT
- for session based services:
  - Without redirect
  - With redirect to top-up-server functionality
  - Screening

In this chapter

This chapter contains the following topics.

Check Balance

Introduction

The Check Balance control plan determines if the user is able to reserve a specified number of units. It returns either a success or failure only; it does not return the number of units in the balance.

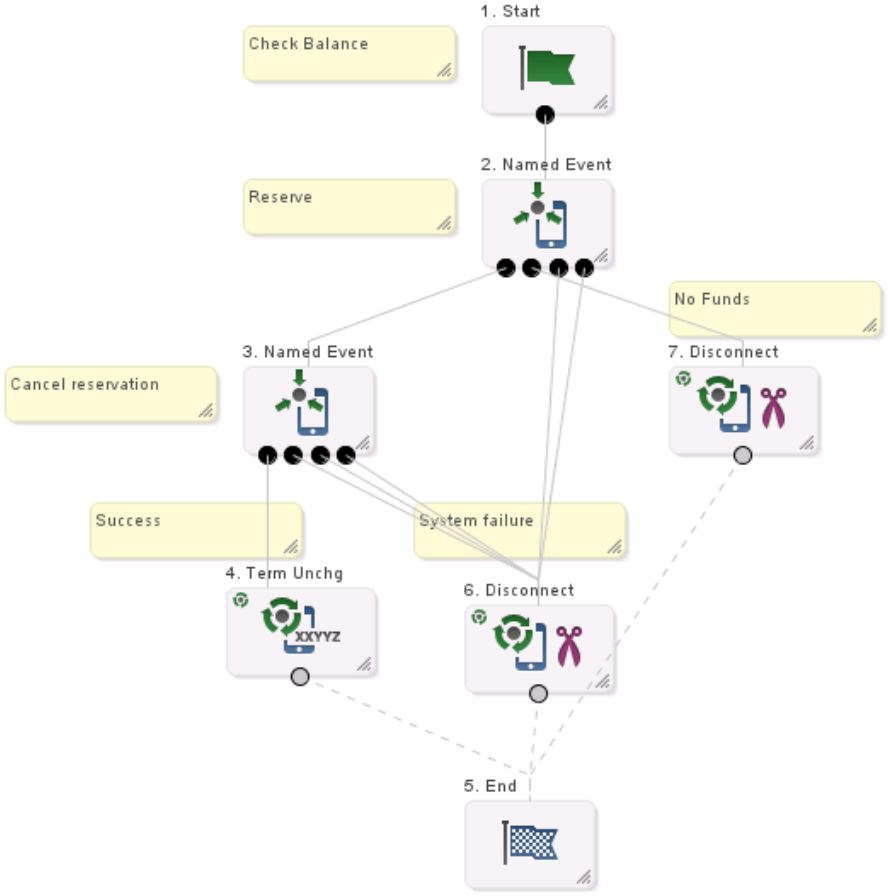
This control plan consists of a start node followed by two Named Event nodes and a terminate unchanged node, with Disconnect nodes as appropriate. The first Named Event node reserves an event type (the `Reserve Event` option selected), appropriate for this service. If the first Named Event node:

- Fails to reserve the event, it goes to a Disconnect node with the reason set to the configured no funds cause.

  Successfully reserves the event, the second Named Event node cancels the reservation (the `Revoke Event` option selected). Then, a Terminate Unchanged node sends an INAP Continue, which signals to diameterControlAgent that the balance check succeeded.

Check Balance control plan

Here is an example Check Balance control plan.



Direct Debiting

Introduction

This control plan starts with two profile branching nodes to determine if this is a time-based direct debit (through INAP extension 502) with an Event-Timestamp AVP (INAP extension 504).

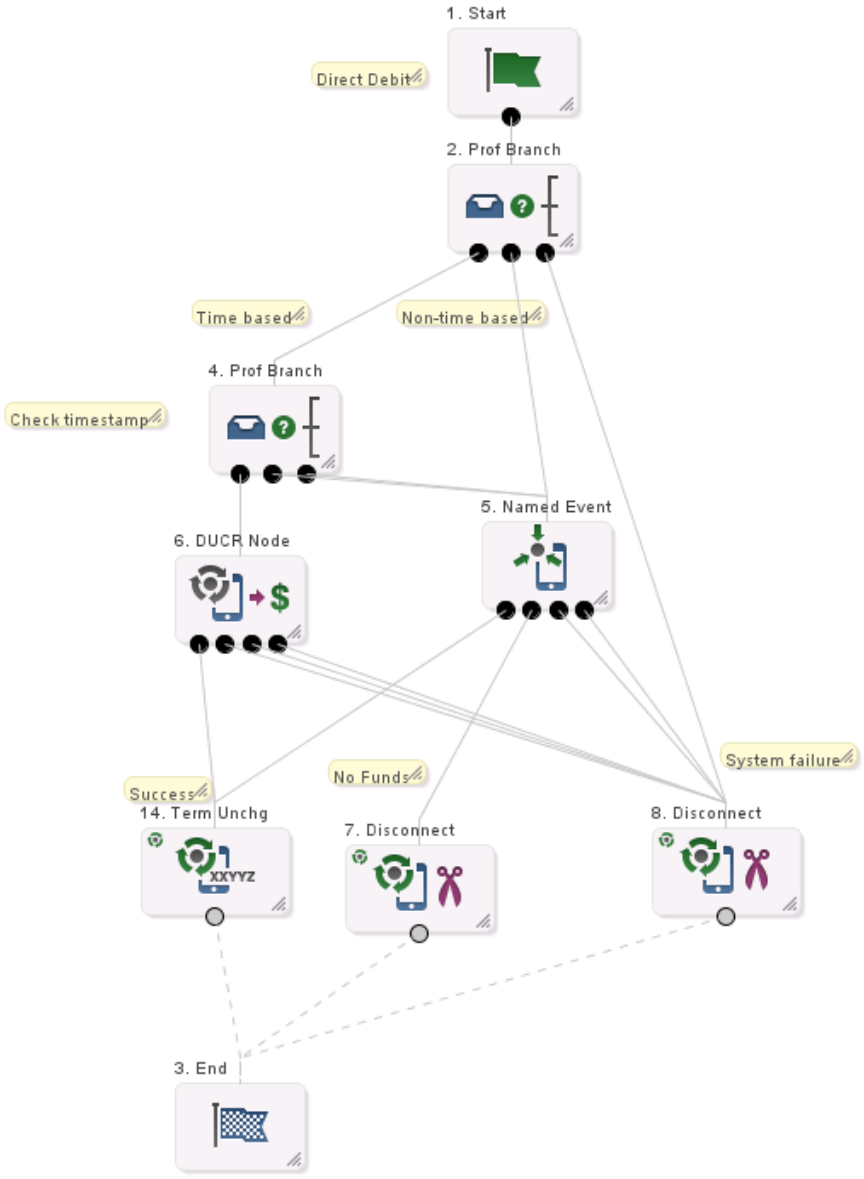
- If it is, a DUCR node is used with the `Debit` option selected to debit the account.
- If it is not, a Named Event node is used with the `Direct Event` option selected to debit the account. The Named Event node reads its number of events from INAP extension 501 (Requested-Service-Units).

Failure branches are connected to Disconnect nodes with appropriate cause values to produce the correct Diameter Result-Code values.

Refer to *INAP Extensions* (on page 13) for details.

Direct Debiting control plan

Here is an example Direct Debiting control plan.



Price Enquiry

Introduction

This control plan has a Named Event node connected to:

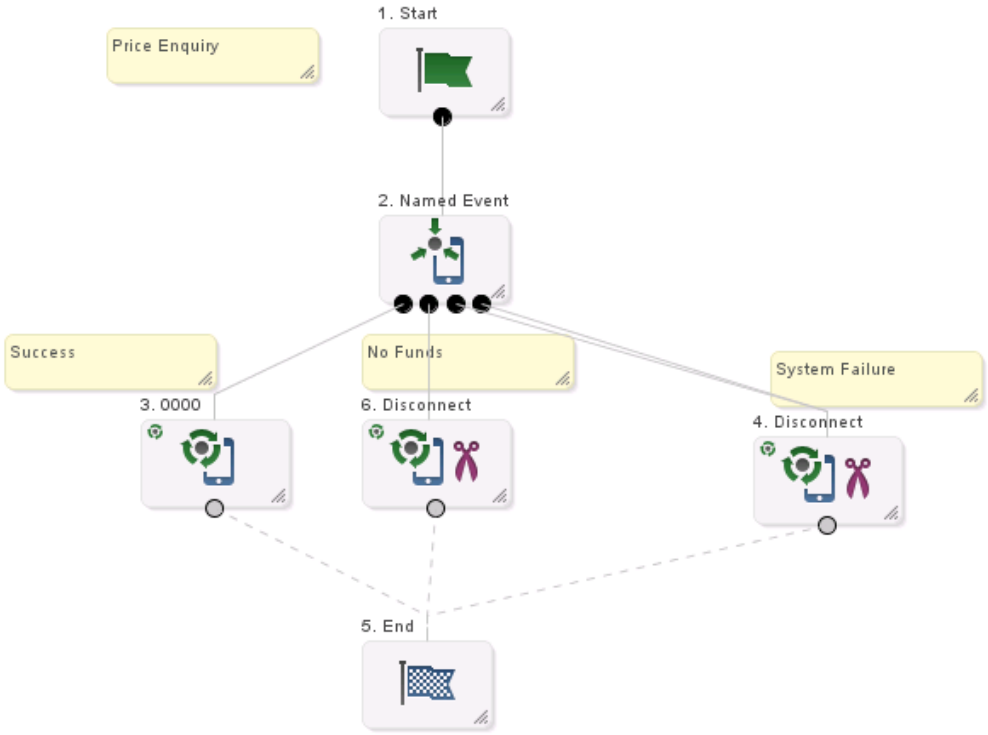
- Disconnect nodes (for failures)
- An unconditional terminate node (for successes)

The Named Event node has the `Cost of event` option selected and is configured to store the cost of the event under a tag in the ACS temporary storage area. Then, the DCA service loader plug-in picks up this tag and puts it in INAP extension 603 in the Connect. The diameterControlAgent copies this into the Cost-information AVP.

Refer to *INAP Extensions* (on page 13) for details.

Price Enquiry control plan

Here is an example Price Enquiry control plan.



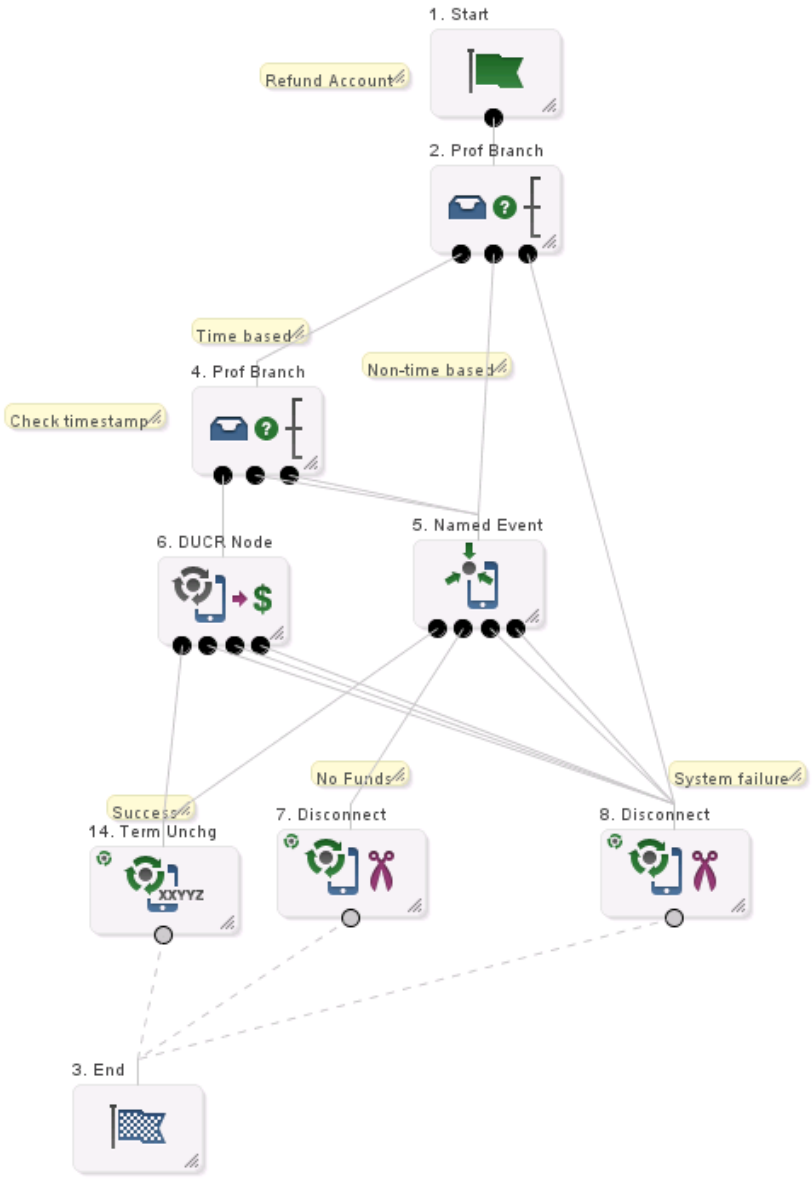
Refund Account

Introduction

The Refund Account control plan is identical to the *Direct Debiting* (on page 124) control plan, except, in the DUCR node, the **Credit** option is selected.

Refund Account control plan

Here is an example Refund Account control plan.



Session No Redirect

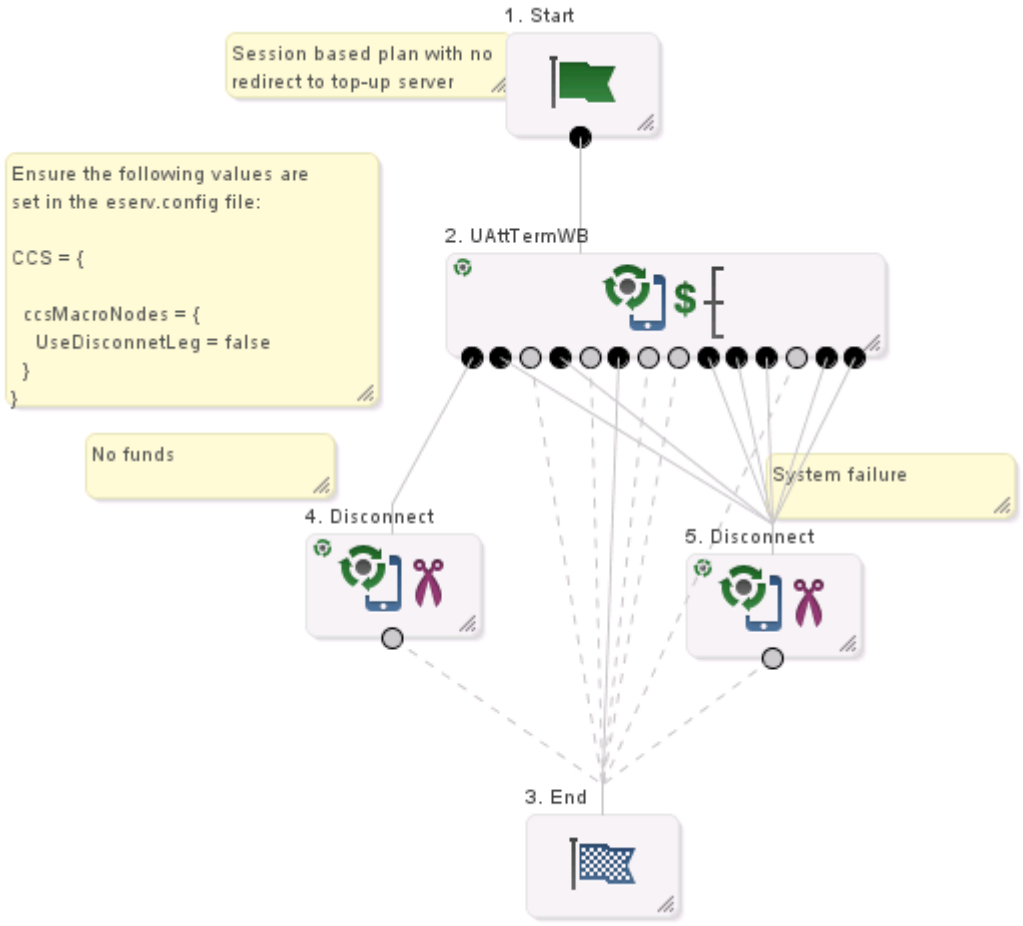
Introduction

The Session No Redirect control plan is a session based plan with no redirect to a top-up server.

This consists of a Start node connected to a UATB node. The exits of the UATB node are connected to an End node (Success cases) and to the Disconnect nodes with various release causes. The release causes in the Disconnect nodes are such as to cause diameterControlAgent to use the appropriate Result-Code.

Session No Redirect control plan

Here is an example Session No Redirect control plan.



eserv.config configuration

As shown in the notes with this control plan, you need to ensure that the following values are set in the `CCS.ccsMacroNodes` section of the **eserv.config** file.

```
CCS = {
    ccsMacroNodes = {
        UseDisconnectLeg = false
    }
}
```

Refer to the *CCS Technical Guide* for details.

Session Redirect

Introduction

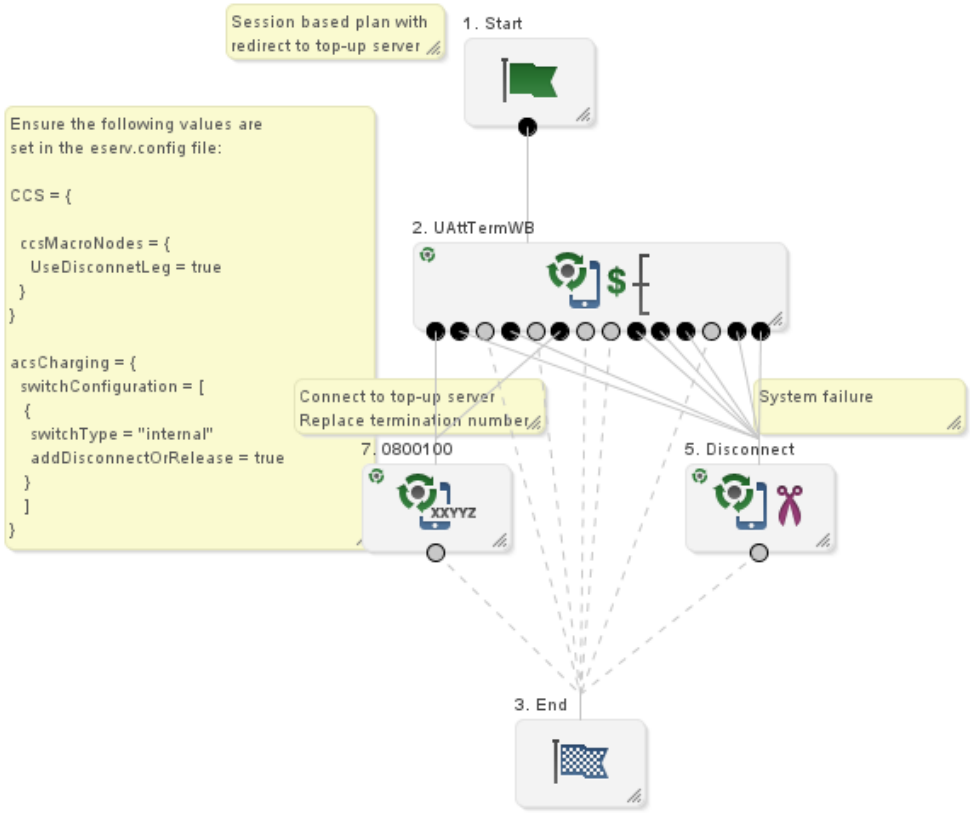
This is the same as the control plan for *Session No Redirect* (on page 127), except the following exits of the UATB node are connected to an unconditional termination node which will contain a number mapped to the address of the top-up-server:

- Declined No Funds

- NSF (Disconnected)

Session Redirect control plan

Here is an example Session Redirect control plan.



eserv.config configuration

As shown in the notes with this control plan, you need to ensure that the following values are set in the `CCS.ccsMacroNodes` section of the **eserv.config** file.

```
CCS = {
    ccsMacroNodes = {
        UseDisconnectLeg = true
    }
}
```

The switch configuration must also be set in the acsCharging.switchConfiguration section.

```
acsCharging = {
    switchConfiguration = [
    {
        switchType = "internal"
        addDisconnectOrRelease = true
    }
    ]
}
```

Refer to *CCS Technical Guide* for details.

Screening

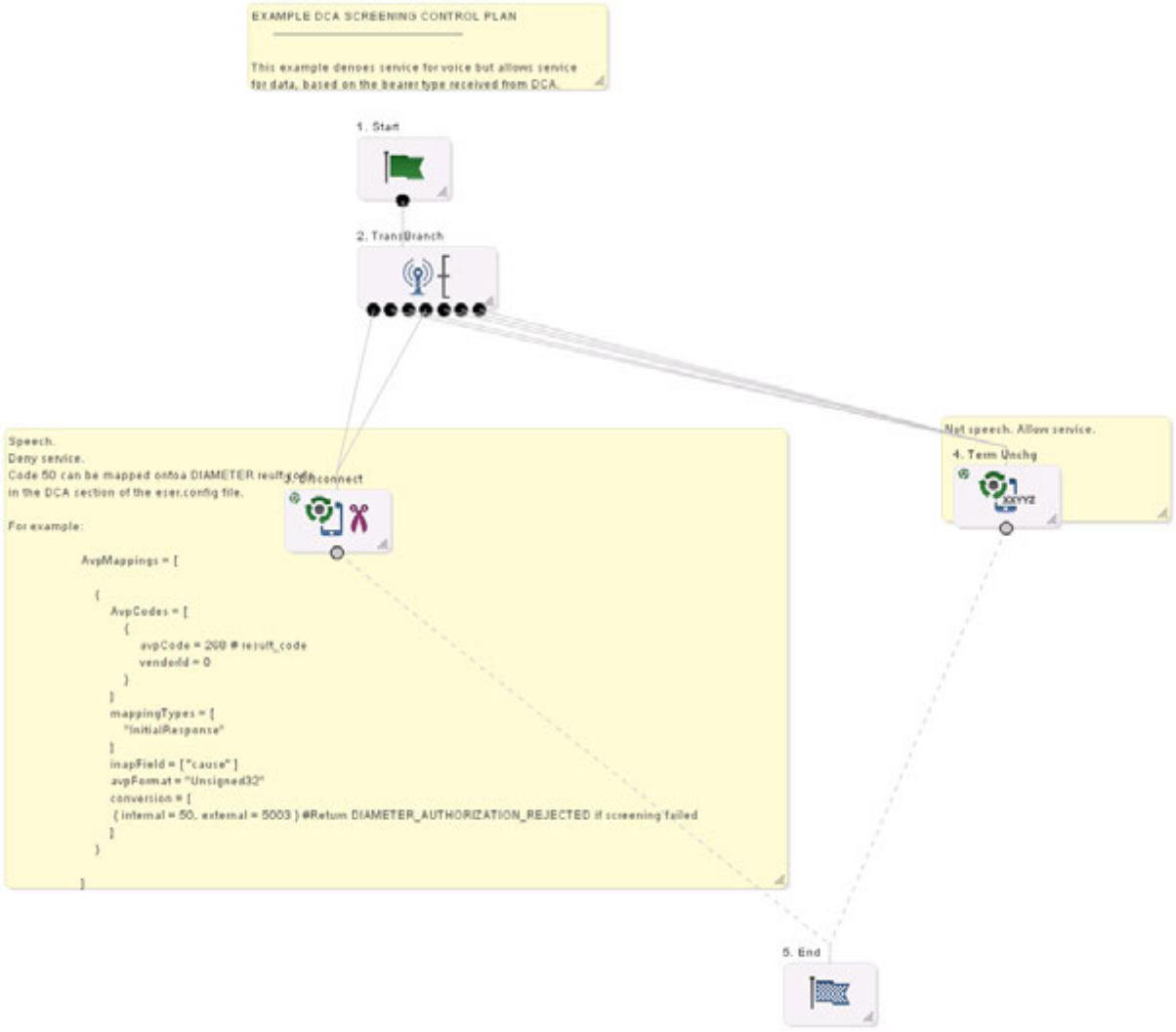
Introduction

The Screening control plan denies service for voice but allows service for data, based on the bearer type received from DCA.

This consists of a Start node connected to a Transmission Type Branch node. The Transmission Type Branch node exits for voice (Exits 1 and 4) are connected to a Disconnect node with a release cause of `50`. The exits for non-voice are connected to a Terminate Uncharged node.

Screening control plan

Here is an example Screening control plan.



eserv.config configuration

As shown in the notes with this control plan, for this example, you need to ensure that the following values are set in the DIAMETER `Services` section of the **eserv.config** file.

```
AvpMappings = [
```

```
    {
        AvpCodes = [
            {
                avpCode = 268 # result_code
                vendorId = 0
            }
        ]
        mappingTypes = [
            "InitialResponse"
        ]
        inapField = [ "cause" ]
        avpFormat = "Unsigned32"
        conversion = [
            { internal = 50, external = 5003 } #Return
            DIAMETER_AUTHORIZATION_REJECTED if screening failed
        ]
    }

]
```

Chapter 6

About Installation and Removal

Overview

Introduction

This chapter provides information about the installed components for the Oracle Communications Network Charging and Control (NCC) application described in this guide. It also lists the files installed by the application that you can check for, to ensure that the application installed successfully.

In this Chapter

This chapter contains the following topics.

Installation and Removal Overview

Introduction

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

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

DCA packages

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

- SMS:
  - dcaSms
- SLC:
  - dcaScp

Checking the Installation

Introduction

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

DCA Scp directories and files

The DCA installation on the SLC creates the following directories:

- **/IN/service_packages/DCA/bin**
- **/IN/service_packages/DCA/etc**

- **/IN/service_packages/DCA/lib**
- **/IN/service_packages/DCA/tmp**

The DCA installation installs the following binaries and interfaces:

- **/IN/services_packages/DCA/bin/diameterControlAgent**

The DCA installation installs the following example configuration file:

- **/IN/services_packages/eserv.config.dca.example**

The DCA installation installs the following shared library:

- **/IN/services_packages/DCA/lib/libdcaCcsSvcExtra.so**

DCA Sms directories

Check that the statistics and control plans have been installed correctly.

The DCA installation on the SMS creates the following directories:

- **/IN/service_packages/DCA/db**
- **/IN/service_packages/DCA/lib**

Chapter 7

Diameter Charging Agent Call Flows

Call Flow Overview

Introduction

This chapter lists a sample set of DCA message flows.

In this chapter

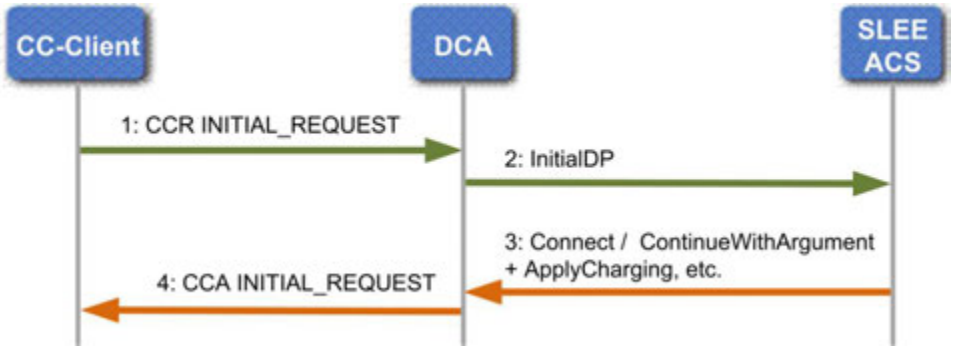
This chapter contains the following topics.

Initial Request Success

Introduction

This example shows the flow for a successful initial request.

Call flow



Comments

This table provides additional comments on the call flow.

Operation	Comment
2	This operation contains a profile block encoded in extension 701, consisting of data mapped from AVPs.

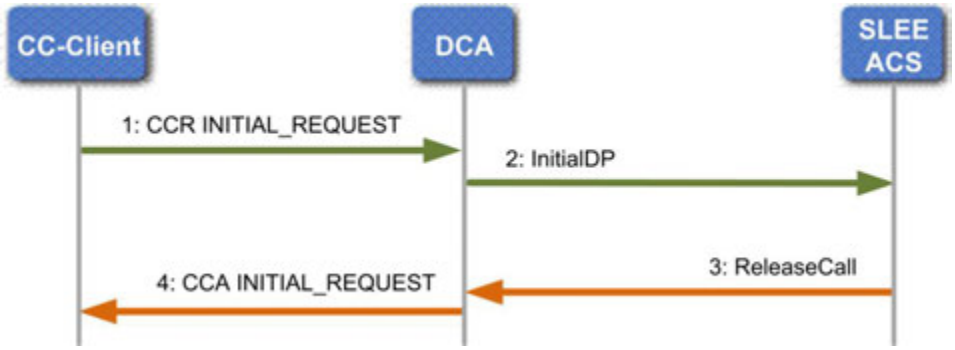
Operation	Comment
3	This operation contains a profile block encoded in extension 701, consisting of data mapped to AVPs.

Initial Request Release Call

Introduction

This example shows the flow for a release call.

Call flow



Comments

This table provides additional comments on the call flow.

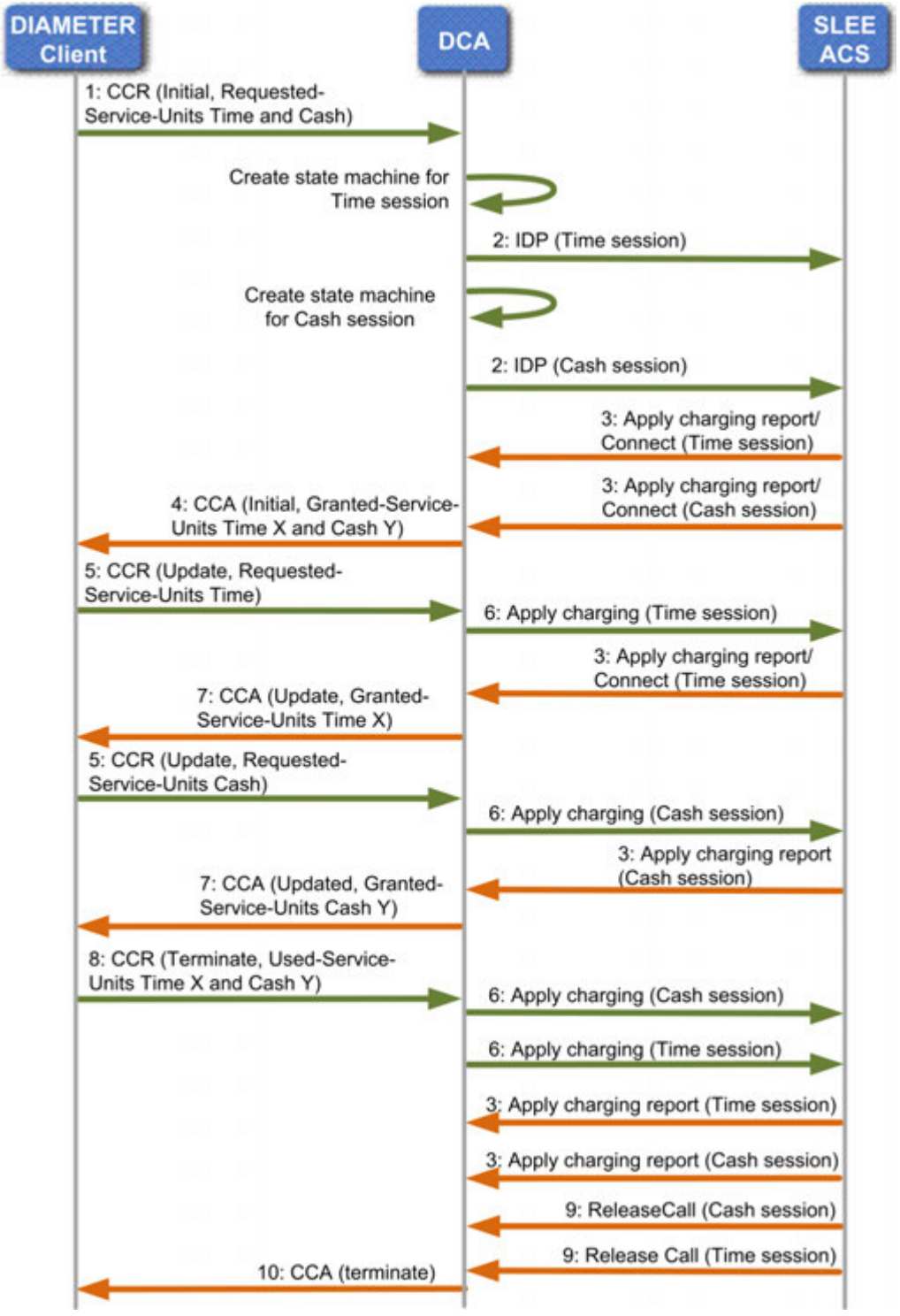
Operation	Comment
2	This operation contains a profile block encoded in extension 701, consisting of data mapped from AVPs.
3	This operation contains a profile block encoded in extension 701, consisting of data mapped to AVPs.

Initial Request Multiple Requested Service Units

Introduction

This example shows the flow when multiple requests for service units (cash or time) are made.

Call flow


DIAMETER Client
DCA
SLEE ACS
1: CCR (Initial, Requested-Service-Units Time and Cash)
Create state machine for Time session
2: IDP (Time session)
Create state machine for Cash session
2: IDP (Cash session)
3: Apply charging report/ Connect (Time session)
3: Apply charging report/ Connect (Cash session)
4: CCA (Initial, Granted-Service-Units Time X and Cash Y)
5: CCR (Update, Requested-Service-Units Time)
6: Apply charging (Time session)
3: Apply charging report/ Connect (Time session)
7: CCA (Update, Granted-Service-Units Time X)
5: CCR (Update, Requested-Service-Units Cash)
6: Apply charging (Cash session)
3: Apply charging report (Cash session)
7: CCA (Updated, Granted-Service-Units Cash Y)
8: CCR (Terminate, Used-Service-Units Time X and Cash Y)
6: Apply charging (Cash session)
6: Apply charging (Time session)
3: Apply charging report (Time session)
3: Apply charging report (Cash session)
9: ReleaseCall (Cash session)
9: Release Call (Time session)
10: CCA (terminate)

AVP Pass-Through DCA to DCD

Introduction

This example shows the flow when a simple pass through from DCA to DCD is made.

DCA Parameters

A large part of configuration for DCA are AVP to profile block definitions.

The following configuration is required to allow an AVP received by DCA from a CC-Client to be passed through to DCD.

- ACS is configured with an Inbound ARRAY-type profile tag, which is used to pass a repeating AVP to ACS.

Note: This may not be necessary if data is not going to be manipulated in a Control Plan, however this would risk having data corrupted if the chosen tag number already exists, and is used in the Control Plan.

- DCA is configured with an inbound mapping from a repeating AVP in a single grouped AVP to an ARRAY-type profile.
- DCD is configured with an outbound mapping from an ARRAY profile tag in the INCOMING_EXTENSIONS profile block.

The following configuration is required to allow an AVP received from a CC-Server by DCD to be passed through to DCA.

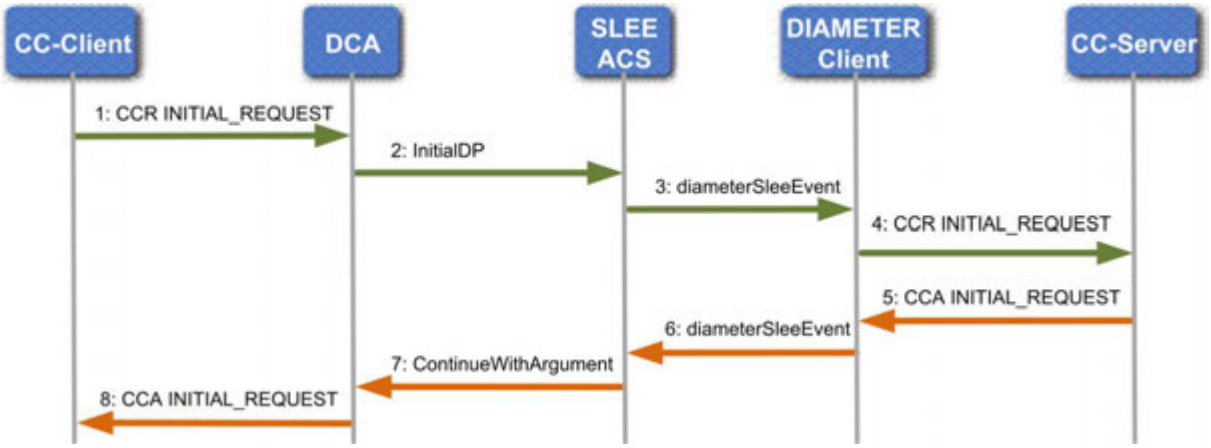
- ACS is configured with an Outbound ARRAY-type profile tag, which is used to pass data from ACS to DCA in the OUTGOING_EXTENSIONS profile block.

Note: This may not be necessary if data is not going to be manipulated in a Control Plan, however this would risk having data corrupted if the chosen tag number already exists, and is used in the control Plan.

- DCD is configured with an inbound mapping to map a repeating AVP in a single grouped AVP to an ARRAY profile in the OUTGOING_EXTENSIONS profile block.
- DCA is configured with an outbound mapping to map an ARRAY profile tag to an AVP.

Call flow

This diagram shows the flow.



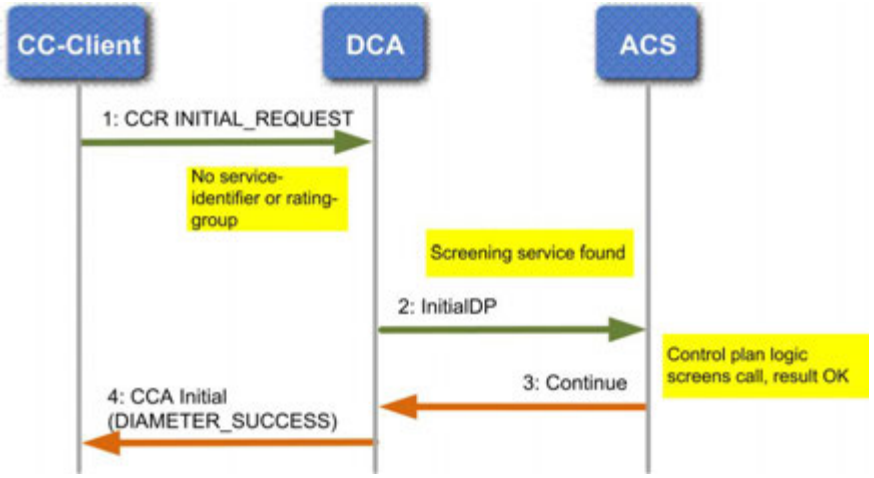
Comments

This table provides additional comments on the call flow.

Operation	Comment
1	CC-Client sends a CCR INITIAL_REQUEST to DCA.
2	DCA sends an InitialDP to ACS containing an ARRAY profile tag in the profile block encoded in the generic_extension_val_extended_os extension (id: 701), in the extensions argument.
3	Upon receipt of the operation, ACS copies the profile tags specified in the generic_extension_val_extended_os into the INCOMING_EXTENSIONS profile block. • The INCOMING_EXTENSIONS profile block is stored in tsMap->incomingExtensionsBlock in acsChassisContext. • In the case of an IDP, the whole generic_extension_val_extended_os extension buffer is re-assigned to tsMap->incomingExtensionsBlock; for an ACR, each profile tag in the extension is individually applied to tsMap->incomingExtensionsBlock.
4	A billing node in the invoked Control Plan results in the DCD actions library being invoked.
5	The DCD actions library: • encodes the ARRAY profile tag found in the INCOMING_EXTENSIONS profile block as an AVP • sends a request in a DiameterSleeEvent to the DCD diameterBeClient.
6	The diameterBeClient sends the CCR request to the CC-Server.
7	CC-Server send a CCA INITIAL_REQUEST.
8	The diameterBeClient sends the response encoded in a DiameterSleeEvent back to ACS

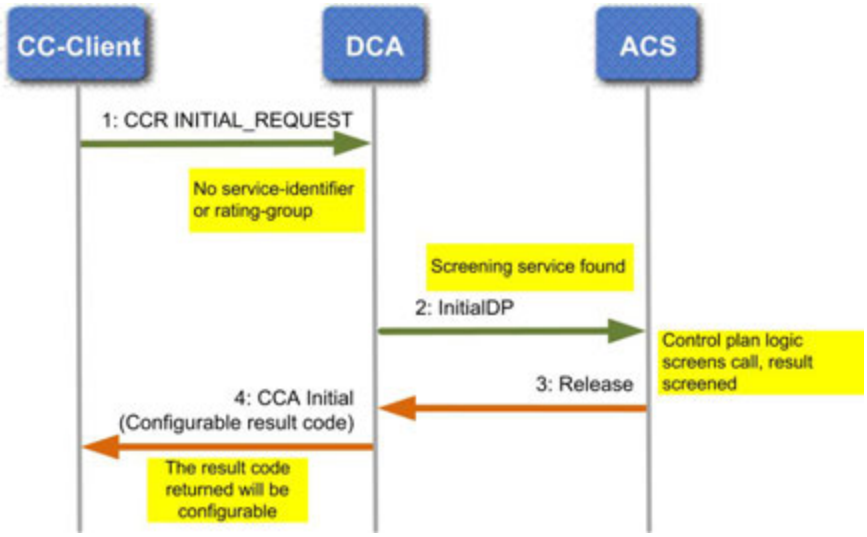
Screening Successful

Call flow



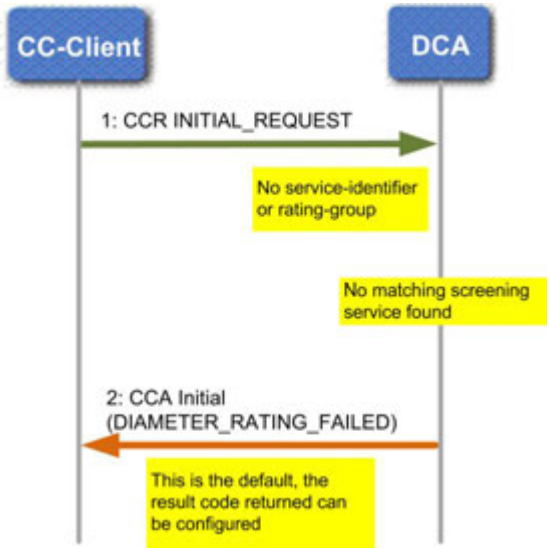
Screening Call Disallowed

Call flow



Screening Failure

Call flow

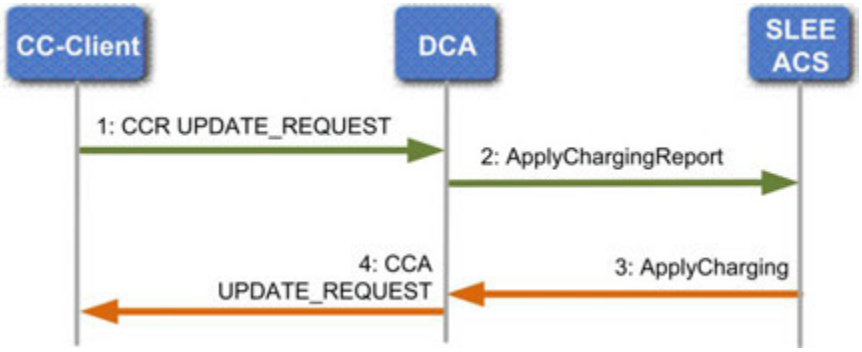


Comments

The call flow shows DIAMETER_RATING_FAILED being returned. This is the default, however the actual returned Result-Code is configurable.

Update Request

Call flow



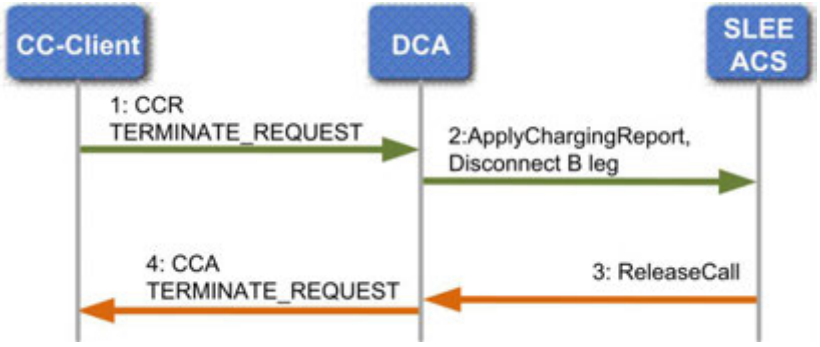
Comments

This table provides additional comments on the call flow.

Operation	Comment
2	This operation contains a profile block encoded in extension 701, consisting of data mapped from AVPs.
3	This operation contains a profile block encoded in extension 701, consisting of data mapped to AVPs.

Terminate Request

Call flow



Comments

This table provides additional comments on the call flow.

Operation	Comment
2	This operation contains a profile block encoded in extension 701, consisting of data mapped from AVPs.
3	This operation contains a profile block encoded in extension 701, consisting of data mapped to AVPs.

Glossary of Terms

AAA

Authentication, Authorization, and Accounting. Specified in Diameter RFC 3588.

ACS

Advanced Control Services configuration platform.

API

Application Programming Interface

AVP

Attribute Value Pair, used in Diameter to represent properties of a particular request or answer.

Call Flow

The path that a caller into the system takes to perform various functions. Often represented diagrammatically as a flowchart or tree-like structure. This shows prompts that the user will hear, with decision points or menus represented by call states.

CAMEL

Customized Applications for Mobile network Enhanced Logic

This is a 3GPP (Third Generation Partnership Project) initiative to extend traditional IN services found in fixed networks into mobile networks. The architecture is similar to that of traditional IN, in that the control functions and switching functions are remote. Unlike the fixed IN environment, in mobile networks the subscriber may roam into another PLMN (Public Land Mobile Network), consequently the controlling function must interact with a switching function in a foreign network. CAMEL specifies the agreed information flows that may be passed between these networks.

CAP

CAMEL Application Part

CC

Country Code. Prefix identifying the country for a numeric international address.

CCA

Credit-Control-Answer, used in Diameter by the credit-control server to acknowledge a Credit-Control-Request (CCR) from the credit-control client.

CCR

Credit-Control-Request, used in Diameter by the credit-control client to request credit authorization from the credit-control server.

CCS

1) Charging Control Services component.

2) Common Channel Signalling. A signalling system used in telephone networks that separates signalling information from user data.

CDR

Call Data Record

Note: The industry standard for CDR is EDR (Event Detail Record).

Connection

Transport level link between two peers, providing for multiple sessions.

DAP

Data Access Pack. An extension module for ACS which allows control plans to make asynchronous requests to external systems over various protocols including XML and LDAP.

Diameter

A feature rich AAA protocol. Utilises SCTP and TCP transports.

DP

Detection Point

DTMF

Dual Tone Multi-Frequency - system used by touch tone telephones where one high and one low frequency, or tone, is assigned to each touch tone button on the phone.

FDA

First Delivery Attempt - the delivery of a short message directly to the SME rather than relaying it through the MC.

GPRS

General Packet Radio Service - employed to connect mobile cellular users to PDN (Public Data Network- for example the Internet).

GSM

Global System for Mobile communication.

It is a second generation cellular telecommunication system. Unlike first generation systems, GSM is digital and thus introduced greater enhancements such as security, capacity, quality and the ability to support integrated services.

HLR

The Home Location Register is a database within the HPLMN (Home Public Land Mobile Network). It provides routing information for MT calls and SMS. It is also responsible for the maintenance of user subscription information. This is distributed to the relevant VLR, or SGSN (Serving GPRS Support Node) through the attach process and mobility management procedures such as Location Area and Routing Area updates.

HPLMN

Home PLMN

HTML

HyperText Markup Language, a small application of SGML used on the World Wide Web.

It defines a very simple class of report-style documents, with section headings, paragraphs, lists, tables, and illustrations, with a few informational and presentational items, and some hypertext and multimedia.

HTTP

Hypertext Transport Protocol is the standard protocol for the carriage of data around the Internet.

IDP

INAP message: Initial DP (Initial Detection Point)

IMS

IP Multimedia Subsystem (3GPP) enables the use of multimedia services based on and built upon Internet applications, services and protocols. These protocols include SIP, which is used to manage the IP multimedia sessions.

IMSI

International Mobile Subscriber Identifier. A unique identifier allocated to each mobile subscriber in a GSM and UMTS network. It consists of a MCC (Mobile Country Code), a MNC (Mobile Network Code) and a MSIN (Mobile Station Identification Number).

The IMSI is returned by the HLR query (SRI-SM) when doing FDA. This tells the MSC exactly who the subscriber is that the message is to be sent to.

IN

Intelligent Network

INAP

Intelligent Network Application Part - a protocol offering real time communication between IN elements.

Initial DP

Initial Detection Point - INAP Operation. This is the operation that is sent when the switch reaches a trigger detection point.

IP

1) Internet Protocol

2) Intelligent Peripheral - This is a node in an Intelligent Network containing a Specialized Resource Function (SRF).

IP address

Internet Protocol Address - network address of a card on a computer.

ISDN

Integrated Services Digital Network - set of protocols for connecting ISDN stations.

ITU

International Telecommunication Union

LAC

Location Area Code. This is an integer value specified as the third level of detail in the location area information. One LAC contains multiple Cell IDs or SAIs.

MC

Message Centre. Also known as SMSC.

MCC

Mobile Country Code. In the location information context, this is padded to three digits with leading zeros. Refer to ITU E.212 ("Land Mobile Numbering Plan") documentation for a list of codes.

MNC

Mobile Network Code. The part of an international address following the mobile country code (MCC), or at the start of a national format address. This specifies the mobile network code, that is, the operator owning the address. In the location information context, this is padded to two digits with a leading zero. Refer to ITU E.212 ("Land Mobile Numbering Plan") documentation for a list of codes.

MS

Mobile Station

MSC

Mobile Switching Centre. Also known as a switch.

MSIN

Mobile Station Identification Number.

MSISDN

Mobile Station ISDN number. Uniquely defines the mobile station as an ISDN terminal. It consists of three parts; the country code (CC), the national destination code (NDC) and the subscriber number (SN).

MT

Mobile Terminated

NOA

Nature Of Address - a classification to determine in what realm (Local, National or International) a given phone number resides, for the purposes of routing and billing.

PLMN

Public Land Mobile Network

RAA

Session message: Re-Auth Answer

RADIUS

Remote Authentication Dial-In User Service - a system of distributed security that secures remote access to networks and network services against unauthorised access.

RAR

Session message: Re-Auth Request

SCF

Service Control Function - this is the application of service logic to control functional entities in providing Intelligent Network services.

SCP

Service Control Point. Also known as SLC.

SCTP

Stream Control Transmission Protocol. A transport-layer protocol analogous to the TCP or User Datagram Protocol (UDP). SCTP provides some similar services as TCP (reliable, in-sequence transport of messages with congestion control) but adds high availability.

Session

Diameter exchange relating to a particular user or subscriber access to a provided service (for example, a telephone call).

SGML

Standard Generalized Markup Language. The international standard for defining descriptions of the structure of different types of electronic document.

SGSN

Serving GPRS Support Node

SIP

Session Initiation Protocol - a signaling protocol for Internet conferencing, telephony, event notification and instant messaging. (IETF)

SLC

Service Logic Controller (formerly UAS).

SLEE

Service Logic Execution Environment

SME

Short Message Entity - This is an entity which may send or receive short messages. It may be located in a fixed network, a mobile, or an SMSC.

SMS

Depending on context, can be:

- Service Management System hardware platform
- Short Message Service
- Service Management System platform
- NCC Service Management System application

SN

Service Number

SRF

Specialized Resource Function – This is a node on an IN which can connect to both the SSP and the SLC and delivers additional special resources into the call, mostly related to voice data, for example play voice announcements or collect DTMF tones from the user. Can be present on an SSP or an Intelligent Peripheral (IP).

SRI

Send Routing Information - This process is used on a GSM network to interrogate the HLR for subscriber routing information.

SSP

Service Switching Point

TCP

Transmission Control Protocol. This is a reliable octet streaming protocol used by the majority of applications on the Internet. It provides a connection-oriented, full-duplex, point to point service between hosts.

URI

Uniform Resource Identifier.

URL

Uniform Resource Locator. A standard way of specifying the location of an object, typically a web page, on the Internet.

VLR

Visitor Location Register - contains all subscriber data required for call handling and mobility management for mobile subscribers currently located in the area controlled by the VLR.

VWS

Oracle Voucher and Wallet Server (formerly UBE).

XML

eXtensible Markup Language. It is designed to improve the functionality of the Web by providing more flexible and adaptable information identification.

It is called extensible because it is not a fixed format like HTML. XML is a `metalanguage' — a language for describing other languages—which lets you design your own customized markup languages for limitless different types of documents. XML can do this because it's written in SGML.

Index

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

R

S

T

U

V

W

X