Oracle® Communications
Network Charging and Control
Diameter Control Agent Technical Guide
Release 5.0.1

June 2013
Copyright

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish, or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

If this is software or related documentation that is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, the following notice is applicable:

U.S. GOVERNMENT END USERS: Oracle programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, delivered to U.S. Government end users are "commercial computer software" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, use, duplication, disclosure, modification, and adaptation of the programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, shall be subject to license terms and license restrictions applicable to the programs. No other rights are granted to the U.S. Government.

This software or hardware is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications that may create a risk of personal injury. If you use this software or hardware in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure its safe use. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software or hardware in dangerous applications.

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Intel and Intel Xeon are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. AMD, Opteron, the AMD logo, and the AMD Opteron logo are trademarks or registered trademarks of Advanced Micro Devices. UNIX is a registered trademark of The Open Group.

This software or hardware and documentation may provide access to or information on content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services.
Contents

About This Document ........................................................................................................... v
Document Conventions ........................................................................................................ vi

Chapter 1

System Overview ................................................................................................................. 1
  Overview ............................................................................................................................. 1
  What is the Diameter Control Agent ................................................................................ 1

Chapter 2

Configuration ......................................................................................................................... 5
  Overview ............................................................................................................................. 5
  Configuration Overview ................................................................................................... 5
  eserv.config Configuration ............................................................................................. 6
  SLEE.cfg Configuration .................................................................................................... 7
  acs.conf Configuration ..................................................................................................... 8
  Prepaid Charging Configuration ....................................................................................... 8
  Feature Node Configuration .............................................................................................. 11
  INAP Extensions ............................................................................................................. 12
  Parameter Mappings ........................................................................................................ 14
  Business Scenarios .......................................................................................................... 18

Chapter 3

Background Processes ......................................................................................................... 23
  Overview ............................................................................................................................. 23
  diameterControlAgent Process ....................................................................................... 23
  DCADefaults Configuration Section ............................................................................... 24
  DCAInstances Configuration Section .............................................................................. 33
  Services Configuration ..................................................................................................... 54
  PeerSchemes Configuration Section ................................................................................ 70
  Statistics Logged by diameterControlAgent .................................................................... 74

Chapter 4

Service Specific AVP Mappings .......................................................................................... 77
  Overview ............................................................................................................................. 77
  Introduction ....................................................................................................................... 77
  Basic Array ....................................................................................................................... 79
  Key Array ......................................................................................................................... 82
  Array with Conditions ...................................................................................................... 85
  Array with Context ........................................................................................................... 95
  Conditional AVP ............................................................................................................. 100
  Prefix Tree ...................................................................................................................... 108
  Timestamp ......................................................................................................................... 110

Chapter 5

Control Plans ....................................................................................................................... 113
  Overview ............................................................................................................................. 113
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:

- ACS Technical Guide
- CCS Technical Guide
- CCS User's Guide
- SMS Technical Guide
- SMS User's Guide
- SLEE Technical Guide
Document Conventions

Typographical Conventions

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

<table>
<thead>
<tr>
<th>Formatting convention</th>
<th>Type of information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Special Bold</td>
<td>Items you must select, such as names of tabs. Names of database tables and fields.</td>
</tr>
<tr>
<td>Italics</td>
<td>Name of a document, chapter, topic or other publication. Emphasis within text.</td>
</tr>
<tr>
<td>Button</td>
<td>The name of a button to click or a key to press. Example: To close the window, either click <strong>Close</strong>, or press <strong>Esc</strong>.</td>
</tr>
<tr>
<td>Key+Key</td>
<td>Key combinations for which the user must press and hold down one key and then press another. Example: Ctrl+P, or Alt+F4.</td>
</tr>
<tr>
<td>Monospace</td>
<td>Examples of code or standard output.</td>
</tr>
<tr>
<td>Monospace Bold</td>
<td>Text that you must enter.</td>
</tr>
<tr>
<td>variable</td>
<td>Used to indicate variables or text that should be replaced.</td>
</tr>
<tr>
<td>menu option &gt; menu option &gt;</td>
<td>Used to indicate the cascading menu option to be selected, or the location path of a file. Example: <strong>Operator Functions &gt; Report Functions</strong>. Example: /IN/html/SMS/Helpext/</td>
</tr>
<tr>
<td>hypertext link</td>
<td>Used to indicate a hypertext link on an HTML page.</td>
</tr>
</tbody>
</table>

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 the Diameter Control Agent

What is the 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 60) in the Services section
- ACS Control Plan for Screening (on page 120)
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
- eserv.config Configuration
- SLEE.cfg Configuration
- acs.conf Configuration
- Prepaid Charging Configuration
- Feature Node Configuration
- INAP Extensions
- Parameter Mappings
- Business Scenarios

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:

<table>
<thead>
<tr>
<th>Component</th>
<th>Locations</th>
<th>Description</th>
<th>Further Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>eserv.config</td>
<td>all SLC machines</td>
<td>DCA is configured by the DIAMETER section of eserv.config.</td>
<td>eserv.config Configuration (on page 6).</td>
</tr>
<tr>
<td>eserv.config</td>
<td>all SLC machines</td>
<td>DCA services mappings are configured in the ccsServiceLibrary section of eserv.config.</td>
<td>CCS Service Library configuration (on page 8).</td>
</tr>
<tr>
<td>SLEE.cfg</td>
<td>all SLC machines</td>
<td>The SLEE interface is configured to include the DCA service.</td>
<td>SLEE.cfg Configuration (on page 7) and SLEE Technical Guide.</td>
</tr>
</tbody>
</table>
eserv.config Configuration

Introduction

The eserv.config file is a shared configuration file, from which many 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 up 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 may be used, as in this example:

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

or

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

or

```plaintext
{  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 will cause 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 will ensure you have a working copy to which you can return.

Loading eserv.config changes

If you change the configuration file, then 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 8).

Note: The DCA configuration options in eserv.config are explained in the section on the diameterControlAgent background process (on page 23).

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 slex_acs Dca_CB
SERVICE=Dca_PE 1 slex_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.

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 113) 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:

```plaintext
CCS = {
    ...
    ccsServiceLibrary = {
        ...
    }
    ...
```
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.

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
</table>
| 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. |
From the Named Event grid, select **Cent** and click **Edit**.

**Result:** The Edit Named Event screen appears.
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.


### 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*
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 113) 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.

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

The following INAP extensions are used in the Connect operation.

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

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.

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

<table>
<thead>
<tr>
<th>AVP</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Service-identifier or Rating-Group (where no Service-identifier).</td>
</tr>
<tr>
<td></td>
<td>Requests received are handled as follows:</td>
</tr>
<tr>
<td></td>
<td>- INITIAL-REQUEST - one IDP is sent for each service.</td>
</tr>
<tr>
<td></td>
<td>The Diameter session will have multiple INAP dialogs with slee_acs.</td>
</tr>
<tr>
<td></td>
<td>- UPDATE-REQUEST for a new Service-identifier/Rating-Group - starts</td>
</tr>
<tr>
<td></td>
<td>a new service within the session and causes DCA to send another IDP.</td>
</tr>
<tr>
<td></td>
<td>- UPDATE-REQUEST with no requested-service-unit AVP - ends a service</td>
</tr>
<tr>
<td></td>
<td>within the session.</td>
</tr>
<tr>
<td></td>
<td>- TERMINATION-REQUEST - ends the whole session.</td>
</tr>
<tr>
<td>Note:</td>
<td>If more than one unit type is received within this AVP, DCA will</td>
</tr>
<tr>
<td></td>
<td>recognize the used-service-unit AVP in update and termination</td>
</tr>
<tr>
<td></td>
<td>request messages and will extract the relevant unit used. Typically</td>
</tr>
<tr>
<td></td>
<td>this will be the unit previously specified in the granted service unit.</td>
</tr>
<tr>
<td></td>
<td>If no relevant unit is found then DCA returns CCA (Multiple-</td>
</tr>
<tr>
<td></td>
<td>Services-Credit-Control( result-Code = DIAMETER_INVALID_AVP_VALUE )).</td>
</tr>
</tbody>
</table>

| 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.

<table>
<thead>
<tr>
<th>AVP</th>
<th>Set from</th>
</tr>
</thead>
<tbody>
<tr>
<td>Session-Id</td>
<td>The Session-Id AVP of the first message in this transaction. (stored in dca::StateMachine::session_id)</td>
</tr>
<tr>
<td>Result-Code</td>
<td>Set to DIAMETER_SUCCESS unless otherwise stated.</td>
</tr>
<tr>
<td>Origin-Host</td>
<td>The stack code in the DIAMETER module sets this.</td>
</tr>
<tr>
<td>Origin-Realm</td>
<td>The stack code in the DIAMETER module sets this.</td>
</tr>
<tr>
<td>Auth-Application-Id</td>
<td>Set to 4</td>
</tr>
<tr>
<td>CC-Request-Type</td>
<td>Leave as the stack default, that is, the value of CC-Request-Type from</td>
</tr>
<tr>
<td></td>
<td>the corresponding request.</td>
</tr>
<tr>
<td>CC-Request-Number</td>
<td>Leave as the stack default, that is, the value of CC-Request-Number from</td>
</tr>
<tr>
<td></td>
<td>the corresponding request.</td>
</tr>
<tr>
<td>User-Name</td>
<td>Not set</td>
</tr>
<tr>
<td>CC-Session-Failover</td>
<td>Not set (will default to FAILOVER-NOT-SUPPORTED according to RFC 4006)</td>
</tr>
<tr>
<td>CC-Sub-Session-Id</td>
<td>Set to the value from the corresponding request message, of</td>
</tr>
<tr>
<td>AVP</td>
<td>Set from</td>
</tr>
<tr>
<td>----------------------------</td>
<td>--------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Acct-Multi-Session-Id</td>
<td>Set to the value from the corresponding request message, of present.</td>
</tr>
<tr>
<td>Origin-State-Id</td>
<td>Set to <code>dca::ControlAgent::originStateId</code>.</td>
</tr>
<tr>
<td>Event-Timestamp</td>
<td>Set to the value of the Event-Timestamp AVP from the corresponding request.</td>
</tr>
<tr>
<td>Granted-Service-Unit</td>
<td>For session based services, this is <code>ApplyCharging.maxDuration</code> (divided by 10 if the unit type is Time). 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.</td>
</tr>
<tr>
<td>Multiple-Services-Credit-Control</td>
<td>DCA will populate the MSCC AVPs in CCA messages with the following sub-AVPs where applicable:</td>
</tr>
<tr>
<td></td>
<td>• Granted-Units</td>
</tr>
<tr>
<td></td>
<td>• Rating-Group or Service-Identifier</td>
</tr>
<tr>
<td></td>
<td>• Result-Code</td>
</tr>
<tr>
<td></td>
<td>• Time-Quota-Threshold (AVP code 868)</td>
</tr>
<tr>
<td></td>
<td>• Volume-Quota-Threshold (AVP code 869)</td>
</tr>
<tr>
<td></td>
<td>• Validity-Time (if applicable)</td>
</tr>
<tr>
<td></td>
<td>• Final-Unit-Indication (if applicable)</td>
</tr>
<tr>
<td>Note:</td>
<td>Both Time-Quota-Threshold AVP and Volume-Quota-Threshold AVP will be sent with:</td>
</tr>
<tr>
<td></td>
<td>• Vendor_ID 10415</td>
</tr>
<tr>
<td></td>
<td>• Quota-Threshold value 0</td>
</tr>
<tr>
<td>Cost-Information</td>
<td>For Request-Action type PRICE_ENQUIRY, success case, this comes from the value of extension 603 in the INAP Connect. Otherwise, not set.</td>
</tr>
<tr>
<td>Final-Unit-Indication</td>
<td>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:&lt;Connect destinationRoutingAddress&gt;@&lt;Configured SIP host&gt;.</td>
</tr>
<tr>
<td>Check-Balance-Result</td>
<td>This is derived from the type of INAP operation received as described in the Check balance, with a result of enough credit (on page 20) scenario.</td>
</tr>
<tr>
<td>Credit-Control-Failure-Handling</td>
<td>Set to TERMINATE.</td>
</tr>
<tr>
<td>Direct-Debiting-Failure-Handling</td>
<td>Not set. (According to RFC 4006, it will default to TERMINATE_OR_BUFFER).</td>
</tr>
<tr>
<td>Validity-Time</td>
<td>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 21) scenario.</td>
</tr>
<tr>
<td>Redirect-Host</td>
<td>Not set.</td>
</tr>
<tr>
<td>Redirect-Host-Usage</td>
<td>Not set.</td>
</tr>
<tr>
<td>Redirect-Max-Cache-Time</td>
<td>Not set.</td>
</tr>
<tr>
<td>Proxy-Info</td>
<td>The stack code in the DIAMETER module sets this.</td>
</tr>
</tbody>
</table>
AVP | Set from
---|---
Route-Record | Not set at the moment. 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.

<table>
<thead>
<tr>
<th>Client</th>
<th>DCA</th>
<th>Sle_acs</th>
</tr>
</thead>
<tbody>
<tr>
<td>CCR (INITIAL_REQUEST, Requested-Service-Unit not present or CC-Time present, SubscriptionId = &quot;12345678&quot;)</td>
<td>InitialDp(CallingPartyId = &quot;12345678&quot;, BearerCapability.ITC = 0 = speech)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>RRBCSME (busy, no answer, RSF, abandon, disconnect leg 1 (notify), disconnect leg 2 (interrupted))</td>
<td>ApplyCharging (maxDuration = 5990)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Continue</td>
</tr>
<tr>
<td>CCA (Granted-Service-Unit, CC-Time = 599)</td>
<td></td>
<td>star Tcc</td>
</tr>
<tr>
<td>CCR (UPDATE_REQUEST, Used-Service-Unit, CC-Time = 589)</td>
<td></td>
<td>ApplyChargingReport (timeNoTariffSwitch = 5890, callActive = true)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ApplyCharging (maxDuration = 6100)</td>
</tr>
<tr>
<td>CCA (Granted-Service-Unit, CC-Time = 610)</td>
<td></td>
<td>restart Tcc</td>
</tr>
<tr>
<td>CCR (TERMINATION_REQUEST, Used-Service-Unit, CC-Time = 311)</td>
<td>ERBCSM (oDisconnect, leg1)</td>
<td>ApplyChargingReport (timeNoTariffSwitch = 900, callActive = false)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CCA</td>
</tr>
<tr>
<td></td>
<td>Stop Tcc</td>
<td></td>
</tr>
</tbody>
</table>
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.

diameterControlAgent Process 23
DCADefaults Configuration Section 24
DCAInstances Configuration Section 33
Services Configuration 54
PeerSchemes Configuration Section 70
Statistics Logged by diameterControlAgent 74

diameterControlAgent Process

Purpose

The diameterControlAgent executable is a SLE 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.

```plaintext
DIAMETER = {
    DCADefaults = {
        DCADefaults_parameters
    }
    DCAInstances = [  
        # First Instance  
        {  
            NumberRules = [  
                NumberRules_parameters  
            ]  
            DCAInstances_parameters  
        }  
        RedirectNumberMappings = [  
```

```plaintext
]  
}```
{  
  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.

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 28).

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: Not present means no graceful termination.
Notes: 
Example: gracefulTerminationValidityTime = 300
inapServiceKey
Syntax: \texttt{inapServiceKey = value}
Description: The INAP Key value
Type: Integer
Optionality: Mandatory
Allowed: Any 32 bit integer
Example: \texttt{inapServiceKey = 1234}

invalidMessageSequenceResultCode
Syntax: \texttt{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
Default: 5012 [Diameter unable to comply]
Allowed:
Notes: See Part 7.1 of RFC 3588 and Part 9 of RFC 4006 for a list valid codes.
Example: \texttt{invalidMessageSequenceResultCode = 5012}

itc
Syntax: \texttt{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 \texttt{itc} (on page 58) parameter under the Services section.
Type: Integer, or string
Optionality: Optional
Allowed:
Notes: If automatic setting of ITC is required, then this parameter should be absent.
Example:
\begin{verbatim}
\texttt{itc = 16}
\texttt{or}
\texttt{itc = "3.1kHzAudio"}
\texttt{or}
\texttt{itc = 0x10}
\end{verbatim}

mappingTypes
Syntax: \texttt{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 66) parameter under the Services section.
Example: mappingTypes = ["InitialRequest", "UpdateRequest", "EventRequest", "TerminateRequest"]

maxSessionLengthAfterFinalUnitIndicationsSeconds
Syntax: maxSessionLengthAfterFinalUnitIndicationsSeconds = 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: maxSessionLengthAfterFinalUnitIndicationsSeconds = 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: 5012 [Diameter unable to comply]
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: \(\text{validityTime} = \text{seconds}\)
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: \(\text{validityTime} = 30\)

**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(s) for this AVP 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.

<table>
<thead>
<tr>
<th>AVP Format</th>
<th>Extension Type</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>OctetString</td>
<td>Inap Number</td>
<td>not allowed</td>
</tr>
<tr>
<td>Time</td>
<td>Inap Number</td>
<td>not allowed</td>
</tr>
<tr>
<td>String/OctetString</td>
<td>Integer</td>
<td>• must be ASCII digits</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• converts to Integer</td>
</tr>
<tr>
<td>String</td>
<td>Inap Number</td>
<td>Must be Hex digits</td>
</tr>
<tr>
<td>Integer</td>
<td>String/Octets</td>
<td>converts to string</td>
</tr>
<tr>
<td>Time</td>
<td>Integer</td>
<td>Number of seconds since 1 January 1970</td>
</tr>
<tr>
<td>Time</td>
<td>String/Octets</td>
<td>in format YYYYMMDDHH24mmss</td>
</tr>
</tbody>
</table>

**avpCode**

Syntax: \(\text{avpCode} = \text{code}\)
Description: The AVP code for this AVP.
Type: Integer
Optionality: Mandatory
Notes: This parameter is an element of the AvpCodes parameter array.
Example: \(\text{avpCode} = 1234\)

**AvpCodes**

Syntax: \(\text{AvpCodes} = [\text{avpcodes}]\)
Description: The AVP code[s] specifying the AVP.
Type: Array
Optionality: Optional
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 77)

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 63).

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

eexternal

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 32).

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.

```json
{  
  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.

```json
{  
  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.

```json
{  
  AvpCodes = [  
    {  
      avpCode = 2000  
    }  
  ]
```
DCAInstances Configuration Section

Introduction

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

- **DCAInstances parameters** (on page 34)
- **NumberRules parameters** (on page 40)
- **RedirectNumberMappings parameters** (on page 42)
- **Tracing parameters** (on page 43)
- **DiameterServer parameters** (on page 45)
- **Services Configuration** (on page 54)
- **Service Specific AVP Mappings** (on page 77)

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.

```plaintext
DCAInstances = [
    # First Instance
    { 
        NumberRules = [
            NumberRules_parameters
        ]
    }
]

dummyDestination = "0000"
systemCurrencyCode = 978
systemCurrencyExponent = -2
multipleServicesRatingGroup = 0
allowDefaultRatingGroup = false
customDefaultUnits = false
```

**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"`
dontDiscardRatingGroupInResponse = true

SIPDomain = "SIP_Domain"
SIPPrefix = "SIP_prefix"

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: 20
Default: 20

Notes: Requires sessionBasedDuplicateDetection to be set to true.
Example: ccDuplicateStoreSize = 10

customDefaultUnits

Syntax: customDefaultUnits = true|false
Description: This configuration applies only when multiple service credit control (MSCC) is in use. If set to true, DCA does not terminate sessions when processing initial requests or update requests that do not contain a Requested-Service-Units (RSU) AVP.

If set to false, DCA terminates sessions for initial requests or update requests that do not contain a RSU AVP.

Type: Boolean
Optionality: Optional (default used if not set)
Default: false
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: \texttt{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: \texttt{dontDiscardRatingGroupInResponse = false}

ggsnSupportsFinalUnitIndication
Syntax: \texttt{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: \texttt{ggsnSupportsFinalUnitIndication = true}

instanceName
Syntax: \texttt{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 \textit{SLEE.cfg Configuration} (on page 7).
Default: no default
Notes: 
Example: \texttt{instanceName = "dcaIf"}

invalidMessageSequenceResultCode
Syntax: \texttt{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 \textit{RFC 3588} and Part 9 of \textit{RFC 4006} for a list valid codes.
Example: \texttt{invalidMessageSequenceResultCode = 5012}

maxAnswerReorder
Syntax: \texttt{maxAnswerReorder = number}
Description: The maximum number of answers to consider for reordering, before giving up, and answering.
Chapter 3

### maxAnswerReorder

**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

### 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: no default
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
Chapter 3

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 42) 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 41) 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 is 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:  
Mandatory 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: \( \text{destAddress} = ["addr", "addr"] \)
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:
```text
destAddress = [
  "a.b.c.com.0064213",
  "a.b.c.com.0064214"
]
```

destinationAddressAvp
Syntax: \( \text{destinationAddressAvp} = \text{avp} \)
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:
```text
destinationAddressAvp = 1234
```

enabled
Syntax: \( \text{enabled} = \text{true|false} \)
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:
```text
enabled = false
```

origAddress
Syntax: \( \text{origAddress} = ["addr", "addr"] \)
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:
```text
origAddress = [
  "a.b.c.com.0064212",
  "a.b.c.com.0064213",
  "a.b.c.com.0064214"
]
Chapter 3

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.

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
    percentVolumeQuotaThreshold = 80
    commitGrantedOnTerminate = false
    allowMultiServiceIdentifier = false
    finalGrantUnused = false
sessionLimit = 0  
overLimitError = 3004  
counterLogInterval = 0  
sendCreditLimitReachedOnSessionEnd = false  
chargeOnSessionTimeout = true  
sendAbortOnSessionTimeout = true  
commitGrantedOnSessionTimeout = true  
sessionFallbackTcc = 3600
} # End of DiameterServer section

Acct-Application-Id

syntax: 

Description: 

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 = [IDs]

allowMultiServiceIdentifier - 1.0.2-104553

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: 

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
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: \[\text{connectionTimeout} = \text{seconds}\]

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: \(\text{connectionTimeout} = 30\)

counterLogInterval

Syntax: \[\text{counterLogInterval} = \text{secs}\]

Description: The interval in seconds between sending request counts to the syslog file.

Type: Integer

Optionality: Optional (default used if not set).

Allowed: 0 - Do not log requests.

Default: 0

Example: \(\text{counterLogInterval} = 0\)

duplicateBytes

Syntax: \[\text{duplicateBytes} = \text{bytes}\]

Description: How many bytes to allocate to the duplicate detection buffer.

Type: Integer

Optionality: Mandatory

Default: 31457280 (30 MB)

Example: \(\text{duplicateBytes} = 31457280\)

duplicateTime

Syntax: \[\text{duplicateTime} = \text{seconds}\]

Description: How long to hold End-to-End Identifiers, when considering them for potential duplicates.

Type: Integer

Optionality: Mandatory

Allowed: Seconds

Default: 240

Example: \(\text{duplicateTime} = 240\)

finalGrantUnused

Syntax: \[\text{finalGrantUnused} = \text{true|false}\]

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: \texttt{finalGrantUnused = true}

\textbf{inBufferSize}

\begin{itemize}
\item Syntax: \texttt{inBufferSize = size}
\item Description: The size, in bytes, of inbound transport buffer.
\item Type: Integer
\item Optionality: Mandatory
\item Allowed: \\
\item Default: 0 (kernel default)
\item Example: \texttt{inBufferSize = 16384}
\end{itemize}

\textbf{overLimitError}

\begin{itemize}
\item Syntax: \texttt{overLimitError = int}
\item Description: Sets the error code to use when rejecting a session because the memory or session limit has been exceeded.
\item Type: Integer
\item Optionality: Optional (default used if not set).
\item Allowed: \\
\item Default: 3004 - Diameter to busy
\item Notes: \\
\item Example: \texttt{overLimitError = 3004}
\end{itemize}

\textbf{protocol}

\begin{itemize}
\item Syntax: \texttt{protocol = "protocol"}
\item Description: The protocol for this server.
\item Type: String
\item Optionality: Mandatory
\item Allowed: \\
\item Default: "tcp"
\item Example: \texttt{protocol = "tcp"}
\end{itemize}

\textbf{sctpBindAddress}

\begin{itemize}
\item Syntax: \texttt{sctpBindAddress = "addr"}
\item Description: The SCTP port to listen on for this instance.
\item Type: String
\item Optionality: Mandatory
\item Default: 0 (that is, INADR_ANY)
\item Example: \texttt{sctpBindAddress = "192.168.1.2"}
\end{itemize}

\textbf{sctpListenPort}

\begin{itemize}
\item Syntax: \texttt{sctpListenPort = "port"}
\item Description: The SCTP port to listen on
\item Type: String
\end{itemize}
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"

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: \( \text{sessionFallbackTcc} = \text{seconds} \)

Description: The session fallback tcc timer (in seconds).

Type: Integer

Optionality: Optional (default used if not set).

Allowed: 3600

Default: 3600

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

Example: \( \text{sessionFallbackTcc} = 3600 \)

Supported-Vendor-Id

Syntax: \( \text{Supported-Vendor-Id} = [ \text{values} ] \)

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: \( \text{Supported-Vendor-Id} = [ 16247, 10415 ] \)

thirtyTwoBitQuotaThresholds

Syntax: \( \text{thirtyTwoBitQuotaThresholds} = \text{true|false} \)

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: \( \text{thirtyTwoBitQuotaThresholds} = \text{true} \)

Vendor-Specific-Application-Identifier

Syntax: \( \text{Vendor-Specific-Application-Identifier} = [ \text{values} ] \)

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

Type: Array

Optionality: Optional

Allowed:

Default:

Notes:
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>"
rankingGroup = "<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
    }
See *Service Specific AVP Mappings* (on page 77) 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 60)
- typeCriteria (on page 69)
- the base *Service Specific AVP Mappings* (on page 77)
- within AVP mappings of each level of a nested array

Example:

```json
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 77).

**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. Multiply the value received from ACS by this to get Granted-Device-Units. Divide Used-Service-Units by this value before sending 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
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
Chapter 3

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
Chapter 3, Background Processes

Allowed values are:

<table>
<thead>
<tr>
<th>String</th>
<th>Integer</th>
<th>Hex</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;speech&quot;</td>
<td>0</td>
<td>0x00</td>
<td>Speech</td>
</tr>
<tr>
<td>&quot;udi&quot;</td>
<td>8</td>
<td>0x08</td>
<td>Unrestricted Digital Information</td>
</tr>
<tr>
<td>&quot;rdi&quot;</td>
<td>9</td>
<td>0x09</td>
<td>Restricted Digital Information</td>
</tr>
<tr>
<td>&quot;3.1kHzAudio&quot;</td>
<td>16</td>
<td>0x10</td>
<td>3.1 kHz audio</td>
</tr>
<tr>
<td>&quot;udiTA&quot; or &quot;7kHzAudio&quot;</td>
<td>17</td>
<td>0x11</td>
<td>Unrestricted Digital Information with tones/announcements</td>
</tr>
<tr>
<td>&quot;video&quot;</td>
<td>24</td>
<td>0x18</td>
<td>Video</td>
</tr>
</tbody>
</table>

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: \texttt{requestedAction = "DIRECT\_DEBITING"}

\textbf{screeningService}

\textbf{Syntax:} \texttt{screeningService = true|false}

\textbf{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.

\textbf{Type:} Boolean

\textbf{Optionality:} Optional (default used if not set).

\textbf{Allowed:} true, false

\textbf{Default:} false

\textbf{Notes:} If the \texttt{screeningService} flag is true, then \texttt{serviceIdentifier} and \texttt{ratingGroup} should be blank.

\textbf{Example:} \texttt{screeningService = false}

\textbf{SelectionAVPs}

\textbf{Syntax:} \texttt{SelectionAVPs = [selection\_avps]}

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

\textbf{Type:} Array

\textbf{Optionality:} Optional

\textbf{Allowed:}

\textbf{Default:}

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

If \texttt{SelectionAVPs} is specified then none of \texttt{serviceContextId}, \texttt{serviceIdentifier}, \texttt{ratingGroup}, or \texttt{unitType} can be specified.

\textbf{Example:} \texttt{SelectionAVPs = [}

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

\]

\textbf{selectionAVPsIsChargingKey}

\textbf{Syntax:} \texttt{selectionAVPsIsChargingKey = true|false}

\textbf{Description:} Indicates whether this AVP is a charging key.

\textbf{Type:} Boolean

\textbf{Optionality:} Optional (default used if not set).

\textbf{Allowed:} true - incoming UPDATE\_REQUESTS messages matching this rule, for which there is no current session, will start a new session.

\textbf{Default:} false - such messages will be rejected with a Diameter answer with an error result code.

\textbf{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: \texttt{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: \texttt{sleeTimeout = 15}

tcc

Syntax: \texttt{tcc = seconds}

Description: The session supervision timer timeout

Type: Integer

Optionality: Mandatory

Allowed: number of seconds

Default: 3600

Notes: Refer to RFC 4006

Example: \texttt{tcc = 3600}

unitType

Syntax: \texttt{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: \texttt{unitType = "Time"}

validityTime

Syntax: \texttt{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: \texttt{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 28).

**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 95)

**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 30) 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.
Chapter 3

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 \( \text{AVPs} = [\ldots] \).

If used inside \( \text{AVPs} = [\ldots] \) it must be used as part of the \text{typeCriteria} section. See \text{typeCriteria} (on page 69) for an example.

Example: 
Example 1
\[
\text{excludeIfMatches} = [101, 105]
\]
Example 2
\[
\text{excludeIfMatches} = [\text{"SPAM"}]
\]

\text{inapField}

Syntax: \text{inapField} = [ \text{field1}, \text{field2}, \ldots \]
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 \text{acsProfile} mapping (that is, \text{extensionFormat} should not be set to "encoded", \text{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 \text{ACS Buffers} topic in \text{ACS Feature Nodes User's Guide}.

Example: \text{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: 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 69) 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 82).

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 IncludeIfMatches conditional AVP, then the literal will override the mapped value if the original value is found in the IncludeIfMatches 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
- UINTGER
- UINTGER64
- 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).
Chapter 3

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 67).

Default:

```
allowedTypes = ["InitialRequest", "InitialResponse", "UpdateRequest", "EventRequest", "TerminateRequest"]
```

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:

```
allowedTypes = ["InitialRequest", "UpdateRequest", "EventRequest", "TerminateRequest"]
```

Mapping categories

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

<table>
<thead>
<tr>
<th>Diameter Message</th>
<th>CC-Request-Type</th>
<th>Credit Control</th>
<th>Service</th>
<th>Requested-Service-Unit</th>
<th>Used-Service-Unit</th>
<th>Configuration mappingTypes</th>
</tr>
</thead>
<tbody>
<tr>
<td>CCR</td>
<td>INITIAL_REQ</td>
<td>MSCC</td>
<td>New</td>
<td>New</td>
<td>-</td>
<td>InitialRequest</td>
</tr>
<tr>
<td>CCR</td>
<td>INITIAL_REQ</td>
<td>Basic</td>
<td>New</td>
<td>New</td>
<td>-</td>
<td>InitialRequest</td>
</tr>
<tr>
<td>CCR</td>
<td>UPDATE_REQ</td>
<td>MSCC</td>
<td>New</td>
<td>New</td>
<td>-</td>
<td>InitialRequest</td>
</tr>
<tr>
<td>CCR</td>
<td>UPDATE_REQ</td>
<td>MSCC</td>
<td>Existing</td>
<td>Existing UT</td>
<td>Existing UT</td>
<td>UpdateRequest</td>
</tr>
<tr>
<td>CCR</td>
<td>UPDATE_REQ</td>
<td>Basic</td>
<td>Existing</td>
<td>Existing UT</td>
<td>Existing UT</td>
<td>UpdateRequest</td>
</tr>
<tr>
<td>CCR</td>
<td>UPDATE_REQ</td>
<td>MSCC</td>
<td>Existing</td>
<td>-</td>
<td>Existing UT</td>
<td>TerminateRequest</td>
</tr>
<tr>
<td>CCR</td>
<td>TERM_REQ</td>
<td>MSCC</td>
<td>any</td>
<td>-</td>
<td>-</td>
<td>TerminateRequest</td>
</tr>
<tr>
<td>CCR</td>
<td>TERM_REQ</td>
<td>Basic</td>
<td>any</td>
<td>-</td>
<td>-</td>
<td>TerminateRequest</td>
</tr>
<tr>
<td>CCR</td>
<td>EVENT_REQ</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>EventRequest</td>
</tr>
</tbody>
</table>
Here are the abbreviations used in the table.

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CCR</td>
<td>Credit-Control-Request</td>
</tr>
<tr>
<td>INITIAL_REQ</td>
<td>INITIAL_REQUEST</td>
</tr>
<tr>
<td>UPDATE_REQ</td>
<td>UPDATE_REQUEST</td>
</tr>
<tr>
<td>TERM_REQ</td>
<td>TERMINATION_REQUEST</td>
</tr>
<tr>
<td>EVENT_REQ</td>
<td>EVENT_REQUEST</td>
</tr>
<tr>
<td>MSCC</td>
<td>Multiple-Services-Credit-Control</td>
</tr>
<tr>
<td>Basic</td>
<td>Basic Credit-Control</td>
</tr>
<tr>
<td>Existing UT</td>
<td>Existing Unit Type</td>
</tr>
</tbody>
</table>

**octetsLength**

Syntax: \( \text{octetsLength} = \text{num} \)

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: \( \text{octetsStart} = \text{num} \)

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.

\[
\text{octetsStart} = 3 \\
\text{octetsLength} = 0
\]

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

\[
\text{octetsStart} = -3 \\
\text{octetsLength} = 2
\]

**repeating**

Syntax: \( \text{repeating} = \text{true|false} \)

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 110) 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 88).  
See more examples in:  
- Array with Conditions (on page 85)  
- Conditional AVP (on page 100)

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
permittedInstances = 0
reqScctpInboundStreams = 0
reqScctpOutboundStreams = 0
} # 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"
Chapter 3

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

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 45)
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: \textit{reqSctpInboundStreams} = \textit{num}
Description: The number of requested inbound sctp streams.
Type: \textit{Integer}
Optionality: Mandatory
Notes: There is no guarantee you will actually get these.
Example: \textit{reqSctpInboundStreams} = 8

reqSctpOutboundStreams
Syntax: \textit{reqSctpOutboundStreams} = \textit{num}
Description: The number of requested outbound sctp streams.
Type: \textit{Integer}
Optionality: Mandatory
Notes: There is no guarantee you will actually get these.
Example: \textit{reqSctpOutboundStreams} = 8

Example PeerSchemes section

Here is an example PeerSchemes section of the DIAMETER configuration in the \textit{eserv.config} file.

\begin{verbatim}
PeerSchemes = [
    {
        schemeName = "SchemeA"
        Peers = [
            {
                name = "host1"
                protocol = "both"
                permittedOriginHosts = [
                    "host1.realm1.oracle.com"
                ]
                RemoteAddresses = [
                    "192.168.1.10"
                ]
                netmaskBits = 32
                permittedInstances = 0
                reqSctpInboundStreams = 8
                reqSctpOutboundStreams = 8
            } # end of Peer host1
            {
                name = "host2"
                protocol = "sctp"
                permittedOriginHosts = [
                    "host1.realm1.oracle.com"
                ]
            }
        ]
    }
\end{verbatim}
RemoteAddresses = [
    "192.168.1.11"
]
netmaskBits = 32
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
        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)

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

<table>
<thead>
<tr>
<th>Node Name</th>
<th>Statistics ID</th>
<th>Date</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>mtv-tst-scp10</td>
<td>DUPLICATES_DETECTED</td>
<td>17 August 07 00:52</td>
<td>1</td>
</tr>
<tr>
<td>mtv-tst-scp10</td>
<td>DUPLICATES_DETECTED</td>
<td>17 August 07 00:54</td>
<td>1</td>
</tr>
<tr>
<td>mtv-tst-scp10</td>
<td>INITIAL_REQUESTS_RECEIVED</td>
<td>16 August 07 00:02</td>
<td>1</td>
</tr>
<tr>
<td>mtv-tst-scp10</td>
<td>INITIAL_REQUESTS_RECEIVED</td>
<td>16 August 07 03:04</td>
<td>1</td>
</tr>
<tr>
<td>mtv-tst-scp10</td>
<td>INITIAL_REQUESTS_RECEIVED</td>
<td>16 August 07 22:34</td>
<td>1</td>
</tr>
<tr>
<td>mtv-tst-scp10</td>
<td>INITIAL_REQUESTS_ANSWERED</td>
<td>16 August 07 00:52</td>
<td>2</td>
</tr>
<tr>
<td>mtv-tst-scp10</td>
<td>INITIAL_REQUESTS_ANSWERED</td>
<td>17 August 07 00:54</td>
<td>2</td>
</tr>
<tr>
<td>mtv-tst-scp10</td>
<td>INITIAL_REQUESTS_ANSWERED</td>
<td>17 August 07 01:00</td>
<td>1</td>
</tr>
<tr>
<td>mtv-tst-scp10</td>
<td>INITIAL_REQUESTS_RECEIVED</td>
<td>16 August 07 00:02</td>
<td>1</td>
</tr>
<tr>
<td>mtv-tst-scp10</td>
<td>INITIAL_REQUESTS_RECEIVED</td>
<td>16 August 07 03:04</td>
<td>1</td>
</tr>
</tbody>
</table>
### Completed

<table>
<thead>
<tr>
<th>mtv-tst-scp10</th>
<th>INITIAL_REQUESTS_RECEIVED</th>
<th>16 August 07 22:34</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>mtv-tst-scp10</td>
<td>INITIAL_REQUESTS_RECEIVED</td>
<td>17 August 07 00:52</td>
<td>2</td>
</tr>
<tr>
<td>mtv-tst-scp10</td>
<td>INITIAL_REQUESTS_RECEIVED</td>
<td>17 August 07 00:54</td>
<td>2</td>
</tr>
<tr>
<td>mtv-tst-scp10</td>
<td>INITIAL_REQUESTS_RECEIVED</td>
<td>17 August 07 01:00</td>
<td>1</td>
</tr>
<tr>
<td>mtv-tst-scp10</td>
<td>SESSIONS_TIMED_OUT</td>
<td>17 August 07 00:54</td>
<td>1</td>
</tr>
<tr>
<td>mtv-tst-scp10</td>
<td>UPDATE_REQUESTS_ANSWERED</td>
<td>16 August 07 00:02</td>
<td>1</td>
</tr>
<tr>
<td>mtv-tst-scp10</td>
<td>UPDATE_REQUESTS_ANSWERED</td>
<td>16 August 07 03:04</td>
<td>1</td>
</tr>
<tr>
<td>mtv-tst-scp10</td>
<td>UPDATE_REQUESTS_ANSWERED</td>
<td>16 August 07 22:34</td>
<td>1</td>
</tr>
<tr>
<td>mtv-tst-scp10</td>
<td>UPDATE_REQUESTS_ANSWERED</td>
<td>17 August 07 00:52</td>
<td>2</td>
</tr>
<tr>
<td>mtv-tst-scp10</td>
<td>UPDATE_REQUESTS_ANSWERED</td>
<td>17 August 07 00:54</td>
<td>2</td>
</tr>
<tr>
<td>mtv-tst-scp10</td>
<td>UPDATE_REQUESTS_ANSWERED</td>
<td>17 August 07 01:00</td>
<td>1</td>
</tr>
<tr>
<td>mtv-tst-scp10</td>
<td>UPDATE_REQUESTS_RECEIVED</td>
<td>16 August 07 00:02</td>
<td>1</td>
</tr>
<tr>
<td>mtv-tst-scp10</td>
<td>UPDATE_REQUESTS_RECEIVED</td>
<td>16 August 07 03:04</td>
<td>1</td>
</tr>
<tr>
<td>mtv-tst-scp10</td>
<td>UPDATE_REQUESTS_RECEIVED</td>
<td>17 August 07 00:52</td>
<td>2</td>
</tr>
<tr>
<td>mtv-tst-scp10</td>
<td>UPDATE_REQUESTS_RECEIVED</td>
<td>17 August 07 00:54</td>
<td>2</td>
</tr>
<tr>
<td>mtv-tst-scp10</td>
<td>UPDATE_REQUESTS_RECEIVED</td>
<td>17 August 07 01:00</td>
<td>1</td>
</tr>
</tbody>
</table>
Overview

Introduction

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

In this chapter

This chapter contains the following topics.

Introduction 77
Basic Array 79
Key Array 82
Array with Conditions 85
Array with Context 95
Conditional AVP 100
Prefix Tree 108
Timestamp 110

Introduction

Introduction

The AVP mappings within the Services parameter section are organized as shown in Services configuration structure (on page 54).

There are two types of configuration formats available:

- 'classic' format:
  - Base
- 'nested' format. These are configured within an array of format AVPs = [ ]
  - Basic Arrays (see "Basic Array" on page 79)
  - Key Arrays (see "Key Array" on page 82)
  - Array with Conditions (on page 85)
  - Array with Context (on page 95)
  - Conditional AVP (on page 100)
  - Prefix Tree (on page 108)
  - Timestamp (on page 110)

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|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 69) 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.

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

**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
Chapter 4

- 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 68).

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.conf.
In this example, the basic array contains a list of string-type (that is, profileFormat = "STRING") elements.

```json
{  
  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"
        }
      }
    ]  
  ]  
  # 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 configuration 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.

```json
{  
  AVPs = [  
    {  
      AvpCodes = [  
        { avpCode = 1000 }  
      ]  
    }  
  ]  
  AVPs = [  
    {  
      AvpCodes = [  
        { avpCode = 2000 }  
      ]  
    }  
  ]  
  avpFormat = "UTF8String"  
  encodedExtension {  
    profileTag = 9998  
    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:

```json
# 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.

<table>
<thead>
<tr>
<th>Diameter AVP Root Level</th>
<th>1st Level Value</th>
<th>Profile Block Profile (ARRAY)</th>
<th>Sub-Profile Block Tag</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000 2000</td>
<td>&quot;049772056&quot;</td>
<td>8000</td>
<td>1 9998</td>
<td>&quot;049772056&quot;</td>
</tr>
<tr>
<td>2000</td>
<td>&quot;6449016000&quot;</td>
<td></td>
<td>2 9998</td>
<td>&quot;6449016000&quot;</td>
</tr>
<tr>
<td>2000</td>
<td>&quot;08065006000&quot;</td>
<td></td>
<td>3 9998</td>
<td>&quot;08065006000&quot;</td>
</tr>
</tbody>
</table>
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.

```json
{
    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 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 = 2000 }
        ]
    }
    AVPs = [
        { AvpCodes = [
            { avpCode = 3000 }
        ]
    }
}
The configuration for Key Arrays: Outbound is identical to above, except the mapping types are:

```c
# Specify mapping applies OUTBOUND only.
mappingTypes = ["InitialResponse", "UpdateResponse", "EventResponse", "TerminateResponse"]
encodedExtension = {
    profileTag = 8000
    profileFormat = "ARRAY"
}
```
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.

<table>
<thead>
<tr>
<th>Diameter AVP</th>
<th>Profile Block</th>
</tr>
</thead>
<tbody>
<tr>
<td>Root AVP</td>
<td>Profile Block (ARRAY)</td>
</tr>
<tr>
<td>1000</td>
<td>8000</td>
</tr>
<tr>
<td>4400</td>
<td></td>
</tr>
<tr>
<td>4440</td>
<td></td>
</tr>
<tr>
<td>4410</td>
<td>220</td>
</tr>
<tr>
<td>4420</td>
<td>&quot;123123123&quot;</td>
</tr>
<tr>
<td>4430</td>
<td>&quot;Fred&quot;</td>
</tr>
</tbody>
</table>

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.

```json
{
    "Array with Conditions Example"
    AVPs = [ Root-Level AVPs
        {
            AvpCodes = [ 1st Entry in Root-Level AVPs
                {
                    avpCode = 7020
                }
            ]
        AVPs = [ 1st-Level AVPs
            {
                AvpCodes = [ 2nd-Level AVPs
                    { 1st Entry in 2nd-Level AVPs
                        AvpCodes = [
                            { 2nd-level AVP }
                        ]
                    } AVPs = [ 2nd-Level AVPs
                        { 1st Entry in 2nd-Level AVPs
                            AvpCodes = [
                                { 2nd-level AVP }
                            ]
                        }
                    ]
                }
            ]
        }
    }
}
```
Chapter 4

```c
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 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 Criterion  
AVPs = [  
  # Conditional AVPs for 2nd Type Criterion  
  {  
    AvpCodes = [  
      {  
        avpCode = 3001  
        repeating = true  
      }  
    ]  
    avpFormat = "UTF8String"  
    encodedExtension = {  
      profileTag = 80101  
      profileFormat = "STRING"  
    }  
  }  
]  
```

86  NCC Diameter Control Agent Technical Guide
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

```json
AVPs = [
    {
        AvpCodes = [
            { avpCode = 1000 }
        ],
        AVPs = [
            {
                AvpCodes = [                # <--- 1000.2000
                    { 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"
    }
];
```
```
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.

<table>
<thead>
<tr>
<th>Diameter AVP</th>
<th>Profile Block</th>
<th>Profile Block (ARRAY)</th>
<th>Sub-Profile Block</th>
<th>Tag</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000 2000</td>
<td></td>
<td>8000</td>
<td>100</td>
<td>&quot;6449016000&quot;</td>
<td></td>
</tr>
<tr>
<td>3000</td>
<td></td>
<td>3000</td>
<td>1</td>
<td>&quot;6449016000&quot;</td>
<td></td>
</tr>
<tr>
<td>3001 1</td>
<td></td>
<td>3001</td>
<td>1</td>
<td>&quot;6449016000&quot;</td>
<td></td>
</tr>
<tr>
<td>3001 2</td>
<td></td>
<td>3001</td>
<td>1</td>
<td>&quot;6449016000&quot;</td>
<td></td>
</tr>
<tr>
<td>3002</td>
<td></td>
<td>3001</td>
<td>1</td>
<td>&quot;6449016000&quot;</td>
<td></td>
</tr>
<tr>
<td>3001 3</td>
<td></td>
<td>3001</td>
<td>1</td>
<td>&quot;6449016000&quot;</td>
<td></td>
</tr>
<tr>
<td>3001 4</td>
<td></td>
<td>3001</td>
<td>1</td>
<td>&quot;6449016000&quot;</td>
<td></td>
</tr>
<tr>
<td>3001</td>
<td></td>
<td>3001</td>
<td>1</td>
<td>&quot;6449016000&quot;</td>
<td></td>
</tr>
</tbody>
</table>

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.

```json
AVPs = [
    {
        AvpCodes = [
            { avpCode = 1000 }
        ]
    ]
AVPs = [
    {
        AvpCodes = [
            { avpCode = 2000 },
            repeating = true
        ]
    ]
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.

<table>
<thead>
<tr>
<th>Root AVP</th>
<th>1st Level</th>
<th>2nd Level</th>
<th>Value</th>
<th>Profile Block (ARRAY)</th>
<th>Sub-Profile Block</th>
<th>Tag</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>2000</td>
<td>3000</td>
<td>1</td>
<td>8000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>3001</td>
<td>&quot;6449016000&quot;</td>
<td>100</td>
<td>&quot;6449016000&quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>3001</td>
<td>&quot;6449016001&quot;</td>
<td>2</td>
<td>99</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2000</td>
<td>3000</td>
<td>2</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>3001</td>
<td>&quot;123123123&quot;</td>
<td>3</td>
<td>99</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>3001</td>
<td>&quot;6449016000&quot;</td>
<td>100</td>
<td>&quot;6449016001&quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>3001</td>
<td>&quot;6449016001&quot;</td>
<td>100</td>
<td>&quot;6449016002&quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>3001</td>
<td>&quot;6449016000&quot;</td>
<td>100</td>
<td>&quot;6466666666&quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2000</td>
<td>3000</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>3001</td>
<td>&quot;6466666666&quot;</td>
<td>100</td>
<td>&quot;6466666666&quot;</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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.

```json
{ 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"
    }
```
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.

### 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.

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

# Just map as follows (no type Criteria specified)  
AVPs = [  
  {  
    AvpCodes = [  
      { avpCode = 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.

<table>
<thead>
<tr>
<th>Profile Block</th>
<th>Sub-Profile Block</th>
<th>Tag</th>
<th>Value</th>
<th>Diameter AVP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Profile Block (ARRAY)</td>
<td>Sub-Profile Block</td>
<td>Name</td>
<td>Value</td>
<td>Root AVP</td>
</tr>
<tr>
<td>8000</td>
<td>1</td>
<td>100</td>
<td>&quot;8449016000&quot;</td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td></td>
<td>101</td>
<td>&quot;something&quot;</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>102</td>
<td>&quot;else&quot;</td>
<td>2000</td>
</tr>
<tr>
<td></td>
<td></td>
<td>103</td>
<td>&quot;and more&quot;</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>100</td>
<td>&quot;5449016001&quot;</td>
<td>2000</td>
<td>3000</td>
</tr>
<tr>
<td></td>
<td></td>
<td>101</td>
<td>&quot;another&quot;</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>102</td>
<td>&quot;something&quot;</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>103</td>
<td>&quot;more&quot;</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>100</td>
<td>&quot;6666666666&quot;</td>
<td>2000</td>
<td>3000</td>
</tr>
<tr>
<td></td>
<td></td>
<td>101</td>
<td>&quot;and&quot;</td>
<td>2000</td>
</tr>
<tr>
<td></td>
<td></td>
<td>101</td>
<td>&quot;other&quot;</td>
<td></td>
</tr>
</tbody>
</table>

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.

```
{ 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.

<table>
<thead>
<tr>
<th>Profile Block (ARRAY)</th>
<th>Sub-Profile Block</th>
<th>Tag</th>
<th>Value</th>
<th>Diameter AVP</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Root AVP</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1st Level AVP</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2nd Level AVP</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Value</td>
</tr>
<tr>
<td>8000</td>
<td>1</td>
<td>99</td>
<td>1</td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td></td>
<td>100</td>
<td>&quot;5449016000&quot;</td>
<td>2000</td>
</tr>
<tr>
<td></td>
<td></td>
<td>101</td>
<td>&quot;something&quot;</td>
<td>3000</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>99</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>100</td>
<td>&quot;5449016001&quot;</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>101</td>
<td>&quot;something&quot;</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>102</td>
<td>&quot;else&quot;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>99</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>100</td>
<td>&quot;666666666&quot;</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>101</td>
<td>&quot;others&quot;</td>
<td></td>
</tr>
</tbody>
</table>

---

**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  }  
                ]  
            }  
            TypeCriteria = [  
                {  
                    includeIfMatches = [ 1 ]  # <-- Only profileTag(99)==1 are mapped  
                    encodedExtension = {  
                        profileTag = 99  # Array element in sub-profile block  
                        profileFormat = "INTEGER"  
                    }  
                }  
            ]  
            AVPs = [  
                {  
                    AvpCodes = [  
                        {  avpCode = 3001  }  
                    ]  
                    avpFormat = "UTF8String"  
                    encodedExtension = {  
                        profileTag = 100  # Array element in sub-profile block  
                        profileFormat = "STRING"  
                    }  
                }  
            ]  
        }  
    ]  
}  
```
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.

<table>
<thead>
<tr>
<th>Profile Block (ARRAY)</th>
<th>Sub-Profile Block</th>
<th>Tag</th>
<th>Value</th>
<th>Root AVP</th>
<th>1st Level AVP</th>
<th>2nd Level AVP</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>8000</td>
<td></td>
<td>99</td>
<td>1</td>
<td></td>
<td>1000</td>
<td>2000</td>
<td>3001</td>
</tr>
<tr>
<td></td>
<td></td>
<td>101</td>
<td>&quot;6449016000&quot;</td>
<td></td>
<td></td>
<td></td>
<td>&quot;6449016000&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>101</td>
<td>&quot;something&quot;</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>102</td>
<td>&quot;else&quot;</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>99</td>
<td>2</td>
<td></td>
<td>1000</td>
<td>2000</td>
<td>3001</td>
</tr>
<tr>
<td></td>
<td></td>
<td>100</td>
<td>&quot;6449016001&quot;</td>
<td></td>
<td></td>
<td></td>
<td>&quot;6666666666&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>101</td>
<td>&quot;something&quot;</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>102</td>
<td>&quot;else&quot;</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>99</td>
<td>1</td>
<td></td>
<td>1000</td>
<td>2000</td>
<td>3001</td>
</tr>
<tr>
<td></td>
<td></td>
<td>100</td>
<td>&quot;6666666666&quot;</td>
<td></td>
<td></td>
<td></td>
<td>&quot;6666666666&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>101</td>
<td>&quot;other&quot;</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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.

```plaintext
{ # 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 2nd-Level AVPs (Content AVP)
    ] # End of 1st- Level AVPs
} # End of 1st- Level AVPs

{ # End of 2nd- Level AVPs (Context AVP)
    AvpCodes = [
        { avpCode = 2001 # This is AVP
          7030.1000.2000.2001
        }
    ]
    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-Level AVPs

Chapter 4
# 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.

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


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.

<table>
<thead>
<tr>
<th>Diamter Message</th>
<th>Profile Block</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Root AVP</strong></td>
<td><strong>Profile Block (ARRAY)</strong></td>
</tr>
<tr>
<td>1000</td>
<td>8000</td>
</tr>
<tr>
<td>2000</td>
<td>101</td>
</tr>
<tr>
<td>2001</td>
<td>6449016001</td>
</tr>
<tr>
<td>2000</td>
<td>3000</td>
</tr>
<tr>
<td>3001</td>
<td>&quot;ContextC&quot;</td>
</tr>
<tr>
<td>2001</td>
<td>3001</td>
</tr>
</tbody>
</table>

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.

```plaintext
AVPs = [ 
  AvpCodes = [ 
    { avpCode = 1000 }, 
  ] 
] 
```
Chapter 4

```json
{
    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.

<table>
<thead>
<tr>
<th>Profile Block (ARRAY)</th>
<th>Sub-Profile Block</th>
<th>Tag</th>
<th>Value</th>
<th>Root AVP</th>
<th>1st Level AVP</th>
<th>2nd Level AVP</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>8000</td>
<td>1</td>
<td>100</td>
<td>“RootContext”</td>
<td>1000</td>
<td>2000</td>
<td>3000</td>
<td>“ContextB”</td>
</tr>
<tr>
<td></td>
<td></td>
<td>101</td>
<td>“ContextB”</td>
<td></td>
<td></td>
<td>3001</td>
<td>$&quot;6449016000$”</td>
</tr>
<tr>
<td></td>
<td></td>
<td>102</td>
<td>$&quot;6449016000$”</td>
<td></td>
<td></td>
<td>3001</td>
<td>“ContextB”</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>100</td>
<td>“RootContext”</td>
<td>2000</td>
<td>3000</td>
<td>3001</td>
<td>“ContextB”</td>
</tr>
<tr>
<td></td>
<td></td>
<td>101</td>
<td>“ContextB”</td>
<td></td>
<td></td>
<td>3001</td>
<td>$&quot;6449016000$”</td>
</tr>
<tr>
<td></td>
<td></td>
<td>102</td>
<td>$&quot;6449016000$”</td>
<td></td>
<td></td>
<td>3001</td>
<td>“ContextB”</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>100</td>
<td>“RootContext”</td>
<td>2001</td>
<td>3000</td>
<td>3001</td>
<td>“123123123”</td>
</tr>
<tr>
<td></td>
<td></td>
<td>101</td>
<td>“ContextC”</td>
<td></td>
<td></td>
<td></td>
<td>“RootContext”</td>
</tr>
<tr>
<td></td>
<td></td>
<td>102</td>
<td>“123123123”</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Array with Context - outbound example 2

Adding repeating to the AVP 3001 configuration shown in Array with Context - outbound example 1 (on page 98), 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.

<table>
<thead>
<tr>
<th>Profile Block (ARRAY)</th>
<th>Sub-Profile Block</th>
<th>Tag</th>
<th>Value</th>
<th>Root AVP</th>
<th>1st Level AVP</th>
<th>2nd Level AVP</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>8000</td>
<td>1</td>
<td>100</td>
<td>“RootContext”</td>
<td>1000</td>
<td>2000</td>
<td>3000</td>
<td>“ContextB”</td>
</tr>
<tr>
<td></td>
<td></td>
<td>101</td>
<td>“ContextB”</td>
<td></td>
<td></td>
<td>3001</td>
<td>$&quot;6449016000$”</td>
</tr>
<tr>
<td></td>
<td></td>
<td>102</td>
<td>$&quot;6449016000$”</td>
<td></td>
<td></td>
<td>3001</td>
<td>“ContextB”</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>100</td>
<td>“RootContext”</td>
<td>2000</td>
<td>3000</td>
<td>3001</td>
<td>“ContextB”</td>
</tr>
<tr>
<td></td>
<td></td>
<td>101</td>
<td>“ContextB”</td>
<td></td>
<td></td>
<td>3001</td>
<td>$&quot;6449016000$”</td>
</tr>
<tr>
<td></td>
<td></td>
<td>102</td>
<td>$&quot;6449016000$”</td>
<td></td>
<td></td>
<td>3001</td>
<td>“ContextB”</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>100</td>
<td>“RootContext”</td>
<td>2001</td>
<td>3000</td>
<td>3001</td>
<td>“123123123”</td>
</tr>
<tr>
<td></td>
<td></td>
<td>101</td>
<td>“ContextC”</td>
<td></td>
<td></td>
<td></td>
<td>“RootContext”</td>
</tr>
<tr>
<td></td>
<td></td>
<td>102</td>
<td>“123123123”</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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.

```json
{
    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
    
    # 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
}
```
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.

```json
{  
  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
      ]
  ]
}```
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.

<table>
<thead>
<tr>
<th>Diameter AVP</th>
<th>1&lt;sup&gt;st&lt;/sup&gt; Level</th>
<th>Value</th>
<th>Profile Block</th>
</tr>
</thead>
<tbody>
<tr>
<td>Root AVP: 440</td>
<td>1</td>
<td>441</td>
<td>Tag: 99123</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Value: &quot;Value for Type 1&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>442</td>
<td></td>
</tr>
</tbody>
</table>

or

<table>
<thead>
<tr>
<th>Diameter AVP</th>
<th>1&lt;sup&gt;st&lt;/sup&gt; Level</th>
<th>Value</th>
<th>Profile Block</th>
</tr>
</thead>
<tbody>
<tr>
<td>Root AVP: 440</td>
<td>1</td>
<td>441</td>
<td>Tag: 99124</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Value: &quot;Value for Type 2&quot;</td>
</tr>
</tbody>
</table>
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.

<table>
<thead>
<tr>
<th>Diameter AVP</th>
<th>Profile Block</th>
</tr>
</thead>
<tbody>
<tr>
<td>Root AVP</td>
<td>Tag</td>
</tr>
<tr>
<td>440</td>
<td>441</td>
</tr>
<tr>
<td></td>
<td>442</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Diameter AVP</th>
<th>Profile Block</th>
</tr>
</thead>
<tbody>
<tr>
<td>Root AVP</td>
<td>Tag</td>
</tr>
<tr>
<td>440</td>
<td>441</td>
</tr>
<tr>
<td></td>
<td>442</td>
</tr>
</tbody>
</table>

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 example 1 (see "Conditional AVP - inbound example 1" on page 102) 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.

### 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.

```json
{
  AVPs = [
    {
      AvpCodes = [
        {
          avpCode = 440
        }
      ]
    },
    TypeCriteria = [
      {
        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"
    }
  },
  {
    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.

<table>
<thead>
<tr>
<th>Profile Block</th>
<th>Diameter AVP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tag</td>
<td>Value</td>
</tr>
<tr>
<td>99123</td>
<td>&quot;Value for Type 1&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

or

<table>
<thead>
<tr>
<th>Profile Block</th>
<th>Diameter AVP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tag</td>
<td>Value</td>
</tr>
<tr>
<td>99124</td>
<td>&quot;Value for Type 2&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Conditional AVP - outbound example 2
Here is an example outbound conditional AVP configuration in the Services AVP Mappings 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.

```json
{  
  AVPs = [  
    {  
      AvpCodes = [  
        {  
          avpCode = 440  
        }  
      ]  
    }  
  ]  
  TypeCriteria = [  
    {  
      # Criteria for match value [1] In this example, this is NOT the matching entry.  
      includeIfMatches = [ 1 ]  
    }  
  ]  
}  

# This is what to include if match found:  
AVPs = [  
  #Include type sub-AVP (the matching condition)  
  {  
    AvpCodes = [  
      {  
        avpCode = 441  
      }  
    ]  
    avpFormat = "Integer32"  
    encodedExtension = {  
      profileTag = 90001  
      profileFormat = "STRING"  
    }  
  }  
  {  
    AvpCodes = [  
      {  
        avpCode = 442  
      }  
    ]  
    avpFormat = "UTF8String"  
    encodedExtension = {  
      profileTag = 99123  
      profileFormat = "STRING"  
    }  
  }  
]  

{  
  # Criteria for match value [2] In this example, this is the matching entry.  
  includeIfMatches = [ 2 ]  
  encodedExtension = {  
    profileTag = 90001  
    profileFormat = "INTEGER"  
  }  
}  

# This is what to include if match found:  
AVPs = [  
  #Include type sub-AVP (the matching condition)  
  {  
    AvpCodes = [  
```
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.

```json
{
    AVPs = [
        # Root-Level AVPs
        { AvpCodes = [
            { avpCode = 7100
            }
        ]
    ]
}
```

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.

<table>
<thead>
<tr>
<th>Profile Block</th>
<th>Diameter AVP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tag Value</td>
<td>Root AVP</td>
</tr>
</tbody>
</table>
| 90001 2       | 440 | 441 | 2
| 99124 "Value for Type 2" | 442 | "Value for Type 2" |
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
      repeating = true
    }
  ]
  avpFormat = "UTF8String"
]
```
To specify outbound, the mappingTypes are specified as:

```plaintext
# 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.

<table>
<thead>
<tr>
<th>Diameter AVP</th>
<th>Profile Block (Prefix Tree)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Root AVP</td>
<td>Value</td>
</tr>
<tr>
<td>1000</td>
<td>2000 &quot;049772&quot;</td>
</tr>
<tr>
<td></td>
<td>2000 &quot;644901&quot;</td>
</tr>
<tr>
<td></td>
<td>2000 &quot;0800500&quot;</td>
</tr>
<tr>
<td>Profile Tag</td>
<td>Value</td>
</tr>
<tr>
<td>7000</td>
<td>&quot;049772&quot;</td>
</tr>
<tr>
<td></td>
<td>&quot;644901&quot;</td>
</tr>
<tr>
<td></td>
<td>&quot;0800500&quot;</td>
</tr>
</tbody>
</table>

### 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.

```plaintext
{
    mappingTypes = ["InitialRequest", "InitialResponse", "UpdateRequest", "UpdateResponse"]
    avpFormat = "Grouped"
}
Chapter 4

Service Specific AVP Mappings

```python
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.

```python
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 69) parameter description for a list of values.
```

Timestamp example 3

This example places a replying timestamp in the outgoing message.

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

avpFormat = "Integer64"
timestamp = "TIME_NOW64"

AvpCodes = [
    {
        avpCode = 833
        vendorId = 10415
    }
    {
        avpCode = 6
        vendorId = 16247
        repeating = True
    }
]
```
avpCode = 833
vendorId = 10415
}
{
  avpCode = 7
  vendorId = 16247
  repeating = True
}
}
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 113
Direct Debiting 114
Price Enquiry 115
Refund Account 116
Session No Redirect 117
Session Redirect 118
Screening 120

Check Balance

Introduction

The Check Balance control plan is used to check if the user is able to reserve a specified number of units. It will return either a success or failure. It will not return the number of units actually 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 will reserve an event type (the Reserve Event option selected), appropriate for this service. If it:

- Fails to reserve the event, it will go to a Disconnect node (reason = the configured no funds cause)
- Succeeds, then the second Named Event node is used to cancel the reservation (the Revoke Event option selected).
Then, a Terminate Unchanged node is used to send an INAP Continue which signals to diameterControlAgent that the balance check succeeded.

**Check Balance control plan**

Here is an example Check Balance control plan.

![Check Balance control plan diagram](image)

**Direct Debiting**

**Introduction**

This control plan starts with two profile branching nodes which determine if this is a time based direct debit (through INAP extension 502) with an Event-Timestamp AVP (INAP extension 504). In this case a DUCR node is used, with the Debit option selected, to debit the account.

Otherwise, a Named Event node, with Direct Event option selected, is used. The Named Event node reads its number of events from INAP extension 501, that is, the 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 12) 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 (dummy number) for the success case.
Chapter 5

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. The DCA service loader plug-in then picks up this tag and puts it in INAP extension 603 in the Connect. The diameterControlAgent then copies this into the Cost-information AVP.

Refer to INAP Extensions (on page 12) for details.

**Price Enquiry control plan**

Here is an example Price Enquiry control plan.

![Price Enquiry control plan diagram]

**Refund Account**

**Introduction**

The Refund Account control plan is identical to the Direct Debiting (on page 114) 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.

```plaintext
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 117), 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.

```plaintext
Ensure the following values are set in the eserv.config file:

CCS = {
  ccsMacroNodes = {
    UseDisconnectLeg = true
  }
}

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

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

Refer to *CCS Technical Guide* for details.

**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.

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

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

```plaintext
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.

```
AvpMappings = {
    {
        AvpCode = 2436
        valueId = 0
    },
    {
        AvpType = ["SResResponse"
        SResField = ["cause"
        wireFormat = "Unspecified"
        activation = [
            {internal = 50, external = 5000} #return DIAMETER_AUTHORIZATION_REQUEST if screening failed
        ]
}]
```

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.
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 details of the installation and removal process for the application.

In this chapter

This chapter contains the following topics.

Installation and Removal Overview 123
Checking the Installation 123

Installation and Removal Overview

Introduction

For information about the following requirements and tasks, see NCC 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
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 ........................................ 125
- Initial Request Release Call .................................... 126
- Initial Request Multiple Requested Service Units .......... 126
- AVP Pass-Through DCA to DCD ............................... 128
- Screening Successful .......................................... 129
- Screening Call Disallowed ..................................... 130
- Screening Failure ............................................... 130
- Update Request ................................................ 131
- Terminate Request ............................................ 131

Initial Request Success

Introduction
This example shows the flow for a successful initial request.

Call flow

```
1: CCR INITIAL_REQUEST
2: InitialDP
3: Connect / ContinueWithArgument + ApplyCharging, etc.
4: CCA INITIAL_REQUEST
```

Comments
This table provides additional comments on the call flow.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>This operation contains a profile block encoded in extension 701, consisting of data mapped from AVPs.</td>
</tr>
</tbody>
</table>
### Operation

<table>
<thead>
<tr>
<th>Operation</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>This operation contains a profile block encoded in extension 701, consisting of data mapped to AVPs.</td>
</tr>
</tbody>
</table>

### Initial Request Release Call

#### Introduction

This example shows the flow for a release call.

#### Call flow

![Call flow diagram]

#### Comments

This table provides additional comments on the call flow.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>This operation contains a profile block encoded in extension 701, consisting of data mapped from AVPs.</td>
</tr>
<tr>
<td>3</td>
<td>This operation contains a profile block encoded in extension 701, consisting of data mapped to AVPs.</td>
</tr>
</tbody>
</table>

### 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

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)
   6: Apply charging (Time session)
   3: Apply charging report/Connect (Time session)
   6: Apply charging (Cash session)

5: CCR (Update, Requested-Service-Units Time)

5: CCR (Update, Requested-Service-Units Cash)

6: Apply charging (Cash session)

7: CCA (Update, Granted-Service-Units Time X)

7: CCA (Updated, Granted-Service-Units Cash Y)

3: Apply charging report (Cash session)

8: CCR (Terminate, Used-Service-Units Time X and Cash Y)
   6: Apply charging (Cash session)
   6: Apply charging (Time session)

10: CCA (terminate)

9: ReleaseCall (Cash session)

3: Apply charging report (Time session)

9: Release Call (Time session)
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.
Chapter 7

Comments

This table provides additional comments on the call flow.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>CC-Client sends a CCR INITIAL_REQUEST to DCA.</td>
</tr>
<tr>
<td>2</td>
<td>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.</td>
</tr>
</tbody>
</table>
| 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 sends a CCA INITIAL_REQUEST. |
| 8         | The diameterBeClient sends the response encoded in a DiameterSleeEvent back to ACS |

Screening Successful

Call flow

[Diagram of the call flow process]
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.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>This operation contains a profile block encoded in extension 701, consisting of data mapped from AVPs.</td>
</tr>
<tr>
<td>3</td>
<td>This operation contains a profile block encoded in extension 701, consisting of data mapped to AVPs.</td>
</tr>
</tbody>
</table>

Terminate Request

Call flow

Comments
This table provides additional comments on the call flow.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>This operation contains a profile block encoded in extension 701, consisting of data mapped from AVPs.</td>
</tr>
<tr>
<td>3</td>
<td>This operation contains a profile block encoded in extension 701, consisting of data mapped to AVPs.</td>
</tr>
</tbody>
</table>
NCC Glossary of Terms

AAA

ACS
Advanced Control Services configuration platform.

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 (or Prepaid Charging) component.
2) Common Channel Signalling. A signalling system used in telephone networks that separates signalling information from user data.
**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.

**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 SAls.

**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.

Oracle
Oracle Corporation

PLMN
Public Land Mobile Network

RADIUS
Remote Authentication Dial-In User Service - a system of distributed security that secures remote access to networks and network services against unauthorised access.

SCF
Service Control Function - this is the application of service logic to control functional entities in providing Intelligent Network services.
**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**


**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 - 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:

- 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).
Index

A
AAA • 135
About Installation and Removal • 125
About This Document • v
Acct-Application-Id • 47
ACS • 135
acs.conf Configuration • 6, 8, 14
allowDefaultRatingGroup • 35
allowMultiServiceIdentifier - 1.0.2-104553 • 47
Array with Conditions • 72, 79, 87
Array with Conditions - inbound - example 1 • 89, 104
Array with Conditions - Inbound - example 2 • 90
Array with Conditions - outbound - example 1 • 92
Array with Conditions - outbound - example 2 • 93
Array with Conditions - outbound - example 3 • 94
Array with Conditions - outbound - example 4 • 96
Array with Conditions configuration • 87
Array with Context • 64, 79, 97
Array with Context - inbound example • 99
Array with Context - outbound example 1 • 100, 102
Array with Context - outbound example 2 • 102
Array with Context configuration • 97
Audience • v
Auth-Application-Id • 47
AVP • 135
AVP format to Extension type • 28
AVP Pass-Through DCA to DCD • 130
avpCode • 28
AvpCodes • 29, 57
avpFormat • 29
avpMappings • 25, 58
AvpMappings parameters • 25, 28, 64
avpType • 58
avpValue • 58

B
Background Processes • 23
Base example • 79
Basic Array • 79, 81
Basic Array configuration • 82
Business Scenarios • 18

C
Call flow • 127, 128, 129, 130, 131, 132, 133
Call Flow • 135
Call Flow Overview • 127
CAMEL • 135
CAP • 135
CC • 135
CCA • 17, 135
ccDuplicateStoreSize • 35
CCR • 14, 135
CCS • 135
CCS Service Library configuration • 5, 7, 8
chargeOnSessionTimeout • 48
Check Balance • 115
Check Balance control plan • 116
Check balance, with a result of enough credit • 18, 20
Checking the Installation • 125
Comments • 127, 128, 131, 132, 133
commitGrantedOnSessionTimeout • 48
commitGrantedOnTerminate • 48
Conditional AVP • 72, 79, 102
Conditional AVP - inbound example 1 • 104, 106
Conditional AVP - outbound example 1 • 107
Conditional AVP - outbound example 2 • 108
Conditional AVP configuration • 103
Conditional AVP example 1 • 105
Conditional AVP example 3 • 106
Configuration • 5
Configuration components • 5
Configuration file format • 6
Configuration Overview • 5
Connection • 14
connectionTimeout • 49
countAVP • 64
Control Plans • 8, 12, 115
conversion • 30
corversionFactor • 58
Copyright • ii
counterLogInterval • 49
customDefaultUnits • 36

D
DCA components • 3
DCA packages • 125
DCA Parameters • 130
DCA Scp directories and files • 125
DCA SLEE configuration • 7
DCA Sms directories • 126
DCA statistics • 76
DCADefaults Configuration Section • 24
DCADefaults parameters • 25
DCAInstances Configuration Section • 33
DCAInstances configuration structure • 34
DCAInstances parameters • 33, 35
destAddress • 44
destination • 43
destAddressAvp • 45
Diagram • 3
Diameter • 136
Diameter Charging Agent Call Flows • 127
serviceIdentifier • 62
SERVICEKEY • 8
serviceName • 63
Services Configuration • 33, 55
Services configuration structure • 56, 79
Services parameter • 57
Session • 139
Session No Redirect • 119, 120
Session No Redirect control plan • 120
Session Redirect • 120, 121
sessionBasedDuplicateDetection • 39
sessionFallbackTcc • 54
sessionLimit • 51
SGML • 139
Sgsn • 139
Simple conditional • 80, 104
SIP • 139
SIPDomain • 39
SIPPrefix • 39
sipScheme • 32
sipScheme example configurations • 32
SLC • 139
Slee • 139
Slee.cfg Configuration • 5, 7, 23, 37
sleeServiceKey • 27, 63
sleeTimeout • 63
Sme • 139
Sms • 139
Sn • 139
SRF • 139
Sri • 140
Ssp • 140
Startup • 23
Statistics Logged by diameterControlAgent • 76
SubscriptionIdTypes • 39
Successful session-based charging, client
   terminates session • 19
Supported-Vendor-Id • 54
System Overview • 1
systemCurrencyCode • 40
systemCurrencyExponent • 40
systemErrorResultCode • 27, 41

T

tcc • 27, 63
TCP • 140
tcpBindAddress • 51
tcpListenPort • 51
Terminate Request • 133
thirtyTwoBitQuotaThresholds • 54
timestamp • 71, 113
Timestamp • 71, 79, 112
Timestamp example 1 • 112
Timestamp example 2 • 113
Timestamp example 3 • 113
traceDebugLevel • 45

Tracing Parameters • 33, 44
type • 44
typeCriteria • 57, 65, 67, 71, 80
Typographical Conventions • vi
U

unitType • 63
Update Request • 133
URI • 140
URL • 140
V

validityTime • 28, 64
vendorId • 33, 55
Vendor-Specific-Application-Identifier • 55
Vlr • 140
Vws • 140
W

watchdogPeriod • 55
What is the Diameter Control Agent • 1