Oracle® Communications Network Charging and Control

Diameter Charging Driver Technical Guide

Release 15.2

January 2026

Copyright

Contents

Chapter 6

About Installation and Removal 97

About This Document

Scope

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

Audience

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

- RFC 3588 – Diameter Base Protocol
- RFC 4006 – Diameter Credit-Control Application

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

This manual describes system tasks that should only be carried out by suitably trained operators.

Related Documents

The following documents are related to this document:

- *Advanced Control Services Technical Guide*
- *Charging Control Services Technical Guide*
- *Charging Control Services User's Guide*
- *Diameter Charging Driver Alarms Guide*
- *Service Management System Technical Guide*
- *Service Management System User's Guide*
- *Service Logic Execution Environment Technical Guide*

Document Conventions

Typographical Conventions

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

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

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

Chapter 1

System Overview

Overview

Introduction

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

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

In this Chapter

This chapter contains the following topics.

What is the Diameter Charging Driver

Overview

Diameter is a protocol that focuses on network access and accounting. The Diameter base protocol provides the minimum requirements needed for Authentication, Authorization, and Accounting (AAA) (*RFC 3588*). You can extend the base protocol by adding commands or AVPs. *RFC 4006* specifies such an extension for applications that can be used to implement real-time credit-control.

The Diameter Charging Driver (DCD) product provides functionality that allows the Prepaid Charging product to interface with applications using the *RFC 3588* and *RFC 4006* protocol. Typically it is expected that Prepaid Charging will interface with a third-party convergent real-time charging system.

DCD contains several components:

- Diameter protocol stack. Implements the RFC 3588/4006 protocol
- Dynamically loadable library (DLL), **diamActions.so**. Implements the required Prepaid Charging functionality
- Diameter client. Implements the network interface to the Diameter

Diameter Credit Control

The Prepaid Charging product uses the Universal-Attempt-Termination-with-Billing (UATB) node for credit-control of telephony (voice) calls. There are a number of other CCS nodes that also use Diameter credit control actions.

RFC 4006 defines credit-control in the following way:

- Credit-control is a process of checking whether credit is available, credit-reservation, deduction of credit from the end user account when service is completed and refunding of reserved credit that is not used.

The Diameter terminology defines an "interrogation" as the request/answer transaction between the client and server.

RFC 4006 defines session based credit-control as:

- A credit-control process that makes use of several interrogations:
  - The first – Used to reserve money from the user's account and to initiate the process.
  - A possible intermediate – May be needed to request new quota while the service is being rendered.
  - The final – Used to exit the process.

The credit-control server is required to maintain session state for session-based credit-control.

Telephony requires session based credit-control. A new session is created when the CCS product detects that an end-user is trying to establish a new telephony call.

Other nodes may use the DCD to send event based (rather than session based) credit control messages for one-time events, for example, SMS (text message).

Process

Prepaid Charging uses the dcdBeClient (Diameter Charging Driver) to send a first interrogation to the Diameter Server. The server rates the request, reserves a suitable amount of money from the user's account, and returns the corresponding amount of credit resources. Prepaid Charging connects the telephony call and monitors the usage of the granted resources.

Prepaid Charging may send an intermediate interrogation to request a new quota of resources when the granted resources have been consumed. When the telephony call ends, Prepaid Charging sends a final interrogation to inform the Diameter Server of the actual amount of resources used. At this point the session is terminated.

Credit Control Messages

RFC 4006 defines two commands used for credit-control encapsulated in the following messages:

- Credit-Control-Request (CCR). Used by the credit-control client to request credit authorization from the credit-control server.
- Credit-Control-Answer (CCA). Used by the credit-control server to acknowledge a CCR from the credit-control client.

AVPs

A detailed list of AVPs for the CCR and CCA messages is given in *RFC 4006* and copied in the next section of this document. Note the CC-Request-Type – an enumeration with the following values:

- INITIAL_REQUEST – First interrogation
- UPDATE_REQUEST – Intermediate interrogation
- TERMINATION_REQUEST – Final interrogation
- EVENT_REQUEST – Event based (not session based)

Note: DCD can be configured to support certain vendor specific applications that add AVPs to the accounting commands of Diameter base protocol. For more information, see the vendor-specific AVPs under *DCD Parameters* (on page 33).

Attribute Value Pairs

In the Diameter protocol message, parameters are specified as Attribute-Value Pairs (AVPs).

An AVP consists of a Code, Flags, Length, optional Vendor-ID, and Data fields. The AVP Code, combined with the Vendor-ID field, identifies the attribute uniquely. The type (format) of the Data field is implied by the Code and Vendor-ID field combination. The following Data formats are specified:

- OctetString
- Integer32
- Integer64
- Unsigned32
- Unsigned64
- Float32
- Float64
- Address
- Time
- UTF8String
- DiameterIdentity
- DiameterURI
- Enumerated
- Grouped
- GroupedUnitValue

Additional EDR Tags

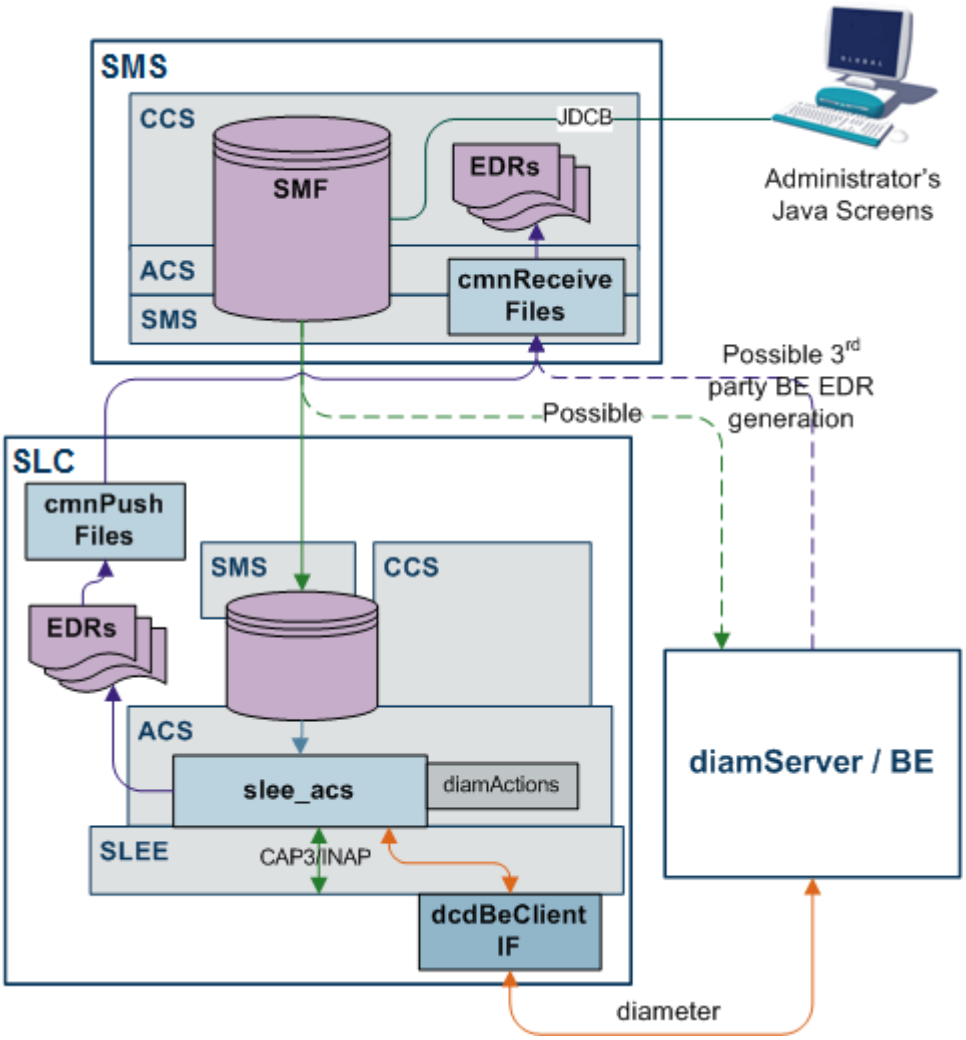
Resolved values for AVPs can be written to the ACS EDR under a configured tag. These tags are not intended to be used to amend existing, predefined ACS tags. The feature is intended for situations where the customer wishes to add some new tag to the EDR.

Conditions can be attached to the writing of the EDR value:

- Replace it unconditionally, after removing any existing tags of same name.
- Append a new value instance unconditionally.
- Leave the EDR alone if the tag is present, and append the new instance if the tag is not present.

Diagram

Here is a high-level diagram showing the Diameter Charging Driver.



ccsConcepts

Introduction

To match AVPs to variables in CCS, the DCD has “ccsConcepts”. These can be a specific parameter of the CCS action, a general CCS variable, some of the call's context, or even an ACS profile value.

The DCD provides functionality to scale values by a factor, and also allows a mapping of one set of integers to another while reading/writing to ccsConcepts. The specific formatting of the value field is configurable. See AVPs parameters for formatting details.

Note: The availability of each concept depends upon the action involved, and the previous actions of the control plan.

List of ccsConcepts

Here is the list of all of the ccsConcepts that can be used in the AVPs section of the **eserv.config** file.

ACS Action Handler

Here are the ccsConcepts from the ACS action handler.

Concept Label	Available	Comments
cascade	After a setCascade Override.	Also can be set by previous responses. The integer ID of the cascade to apply.
chargeInfoBalanceSystemValue	After a chargeInfo response	The balance Unit for the current item of the Charge structure. This is in units of the system currency.
chargeInfoBalanceType	After a chargeInfo response	The CCS ID of the balance Type for the current balance of the Charge structure.
chargeInfoBalanceUnitType	After a chargeInfo response	The CCS of the balance Unit for the current balance of the Charge structure. This is not necessary for a ChargeInfo in a response, it can be derived from the balance type.
chargeInfoBalanceUserValue	After a chargeInfo response	The balance Unit for the current item of the Charge structure. This is in units of the user's currency. Note that the system currency value is mandatory, while this entry is optional.
discountMaxCharge	After a setDiscount	Present after a setDiscount node or a response that has the discountMaxCharge present.
discountPeriod	After a setDiscount	Present after a setDiscount node or a response that has the discountPeriod present.
tariffCugName	After a setTariffPlan	The Closed User Group Name.
tariffPlan	After a setTariffPlan	Integer representing the tariff Plan.
terminationCause	After a call is terminated	The **esg** values in the configuration for the ACS **callEndReasons** that map to specific termination cause values. • 0 = reasonNotSet • 1 = precallAnnouncementFailure • 2 = firstEventACRAbort • 3 = firstEventATAbort • 4 = secondEventACRAbort • 5 = secondEventATAbort • 6 = abortWaitingForBEResponse • 7 = releasedOnTCPExpiry • 8 = releasedNoFunds • 9 = disconnectedLegBNoFunds • 10 = calledPartyBusy • 11 = routeSelectFailure • 12 = callingPartyAbandon • 13 = noAnswer

Concept Label	Available	Comments
		• 14 = callingPartyDisconnected • 15 = calledPartyDisconnected
walletInfoActivationDate	After a walletInfo response	time_t of the wallet's activation date. The DCD handles conversion from time_t to DIAMETER times.
walletInfoBalanceExpiry	After a walletInfo response	The expiry date (in time_t) of the current balance. The DCD handles conversion from time_t to DIAMETER times.
walletInfoBalanceExponent	After a walletInfo response	An exponent to apply to the balance system value.
walletInfoBalanceLimitType	After a walletInfo response	The balances limit type: An integer representing one of: limitedPostpaid, postpaid, prepaid, singleUsePrepaid
walletInfoBalanceMaxCredit	After a walletInfo response	The maximum amount of credit allowed for this subscriber.
walletInfoBalanceSystemValue	After a walletInfo response	The balance Unit for the current item of the balance structure. This is in units of the system currency.
walletInfoBalanceType	After a walletInfo response	The CCS ID of the balance Type for the current balance of the Wallet structure.
walletInfoBalanceUnitType	After a walletInfo response	The CCS ID of the balance Unit for the current balance item of the Wallet structure. This is not necessary for a balance in a response, it can be derived from the balance type.
walletInfoBalanceUserValue	After a walletInfo response	The balance Unit for the current item of the Blance structure. This is in units of the user's currency. Note that the system currency value is mandatory, while this entry is optional.
walletInfoExpiry	After a walletInfo response	The expiry date (in time_t) of the wallet. The DCD handles conversion from time_t to DIAMETER times.
walletInfoLastAccess	After a walletInfo response	time_t of the wallet's last access. The DCD handles conversion from time_t to DIAMETER times.
walletInfoMaxConcurrent	After a walletInfo response	The maximum number of concurrent users allowed for this wallet.
walletInfoState	After a walletInfo response	A single character representing the wallet's state. One of: • 'A' = Active • 'D' = Dormant • 'F' = Frozen • 'P' = Pre-Use • 'S' = Suspended • 'T' = Terminated. Note that conversion to different representations is possible.
walletInfoSystemCurrency	After a walletInfo response	The system currency.

Concept Label	Available	Comments
walletInfoUserCurrency	After a walletInfo response	The CCS_ACCT.CURRENCY value for this wallet.

ACS Service Context

Here are the ccsConcepts from the ACS service context.

Concept Label	Available	Comments
acsCallID	always	The call ID from the SLEE
acsChargingDomain	always	The destined billing domain (logical collection of wallets) for this request.
acsProductType	always	The ACS product type ID
acsProfile	always	An ACS profile buffer from the Call plan. If the buffer is not set, then the AVP is not included.
acsServiceProvider	always	The ACS service provider ID
acsSubscriber	always	The CCS subscriber ID
acsSubscriberReference	always	The CCS subscriber number (ie their MSISDN)
acsTariffCode	After an initial reservation.	Tariff Code string returned in the Initial Reservation Response (if present).
acsUnnormalisedCalledNumber	always	The called party number digits from the IDP, without any attempt at normalization.
acsWallet	always	The CCS wallet ID (BE_WALLET.ID)
acsWalletReference	always	The CCS wallet Reference (the Billing System's reference to the wallet)
acsWalletType	always	The CCS wallet type. (CCS_WALLET_TYPE.ID)

CCS Time Reservation

Here are the ccsConcepts from CCS time reservation.

Concept Label	Available	Comments
callAnsweredTime	ConfirmTimeReservation	
callDurationDelta	Any Time Charging Action	
callDurationTotal	Any Time Charging Action	
callerTimeZone	After a DirectTimeCharge or InitialTimeReservation	

Concept Label	Available	Comments
cli	After a DirectTimeCharge or InitialTimeReservation	
confirmTimeReservationStatus	After set from a response	Usually part of an confirmTimeReservationResponse.
destinationNumber	After a DirectTimeCharge or InitialTimeReservation	
discountPercentage	After a setDiscount or DirectNamedEvent or NamedEventReservation	Present after a setDiscount node or a response that has the discountPeriod present.
eventClass	NamedEvent Actions	A string representing the CCS event Class.
eventName	NamedEvent Actions	A string of the CCS event name.
eventType	NamedEvent Actions	An integer representing the type of CCS named event.
expectedReservationDelta	InitialTimeReservation and ExtendTimeReservation	
expectedReservationTotal	InitialTimeReservation and ExtendTimeReservation	
extraInformation		Usually call information for adding to Billing CDRs. Content varies for each action.
freeCallDisposition	After set from a response	Usually part of an initialTimeReservationResponse.
ignoreBalanceLimit	DirectNamedEvent, DirectTimeCharge, NamedEventReservation	
initialLowBalanceAnnouncement	After set from a response	Usually part of an initialTimeReservationResponse. The Announcement ID of the announcement to play.
initialLowBalanceIndicator	After set from a response	Usually part of an initialTimeReservationResponse. If present and non zero the indicated pre call warning announcement should be played to the subscriber.

Concept Label	Available	Comments
lowCreditBuffer	After set from a response	Usually part of an initialTimeReservationResponse. Number of seconds from the end of the last good reservation period until a low credit beep should be played
maxCallLength	After set from a response	Usually part of an initialTimeReservationResponse.
maxSeconds	After set from a response	Session Time left. Usually part of an xxxTimeReservationResponse.
maxUnitsRequested	NamedEvent Actions	
minUnitsRequested	NamedEvent Actions	
numUnitsGranted	After set from a response	
numUnitsUsed	ConfirmNamedEventReservation	
reservedLengthDelta	After set from a response	Usually part of an xxxTimeReservationResponse.
reservedLengthTotal	After set from a response	Usually part of an xxxTimeReservationResponse.
retrieveLCRNumbers	After set from a response	Usually part of an initialTimeReservationResponse.
revokeTimeReservationStatus	After set from a response	Usually part of an revokeTimeReservationResponse.
scpAction		This AVP is an enumeration with the following known values: • 1 Supervise • 2 Do not supervise • 3 Release • 4 Send message • 5 Play announcement • 6 Supervise without controlling
singleReservation	After set from a response	Usually part of an initialTimeReservationResponse.
timeReservationStatus	After set from a response	Usually part of an xxxTimeReservationResponse.
validityPeriod	After set from a response	

Charge Details

Here are the ccsConcepts from charge details.

Concept Label	Available	Comments
balanceTypeFilter	WalletInfo	Request the Billing Engine to only return balances of this type.

balanceUnitFilter	WalletInfo	Request the Billing Engine to only return balances of this unit.

Direct Time Charge

Here are the ccsConcepts from direct time charge.

Concept Label	Available	Comments
callDate	DirectTimeCharge	
ratingPrecision	InitialTimeReservation	Integer representing seconds, tenths-of-a-second, or hundredths-of-a-second

Others

Here are the ccsConcepts from others.

Concept Label	Available	Comments
freeform	always	Uses/updates the concept previously defined by setFreeform.
setFreeform	always	The next AVP of concept “freeform” will instead use/update the concept indexed by the value of this AVP.

Voucher Details

Here are the ccsConcepts from voucher details.

Concept Label	Available	Comments
voucherInfoBalanceExpiryExtension	WalletInfoRequest	The expiry extension period for adjusting the balance expiry date of the voucher.

<table>
<tr><th>Concept Label</th><th>Available</th><th>Comments</th></tr>
<tr><td>voucherInfoBalanceExpiryExtensionPolicy</td><td>WalletInfoRequest</td><td>Indicates how to apply the balance expiry extension period to the balance expiry date.
New Expiry Policies include the following:
First Use with Offset – Allows a recharged balance expiry date to be set a number of months or hours after the first use of the balance.
First Use – Account Cycle allows a recharged balance expiry date to be set, aligned with the account cycle on the first use of the balance.
First Use – Bill Cycle allows a recharged balance expiry date to be set, aligned with the billing cycle on the first use of the balance.
voucherInfoBalanceExpiryExtensionPolicy returns the following values:<table><tr><th>Value</th><th>Name</th><th>Meaning</th></tr><tr><td>0</td><td>best</td><td>Take the largest expiry date based on current, today, other periods and this extension</td></tr><tr><td>1</td><td>extend</td><td>Extend the existing expiry date by the specified extension period</td></tr><tr><td>2</td><td>extendFromToday</td><td>Today + extension period, or the existing expiry, whichever is larger</td></tr><tr><td>3</td><td>override</td><td>Not used when applying an extension</td></tr><tr><td>4</td><td>dontChange</td><td>Do not set or change an expiry date</td></tr></table></td></tr>
</table>

<table>
<tr><th>Concept Label</th><th>Available</th><th>Comments</th></tr>
<tr><td>voucherInfoBalanceExpiryExtensionType</td><td>WalletInfoRequest</td><td>The unit of the extension value available for this balance (example: hours or months).
voucherInfoBalanceExpiryExtensionType returns the following values:
<table>
<tr><th>Value</th><th>Meaning</th></tr>
<tr><td>0</td><td>Hours</td></tr>
<tr><td>1</td><td>Months</td></tr>
</table></td></tr>
<tr><td>voucherInfoBalanceType</td><td>WalletInfoRequest</td><td>The CCS ID of the balance type for the current balance of the voucher structure.</td></tr>
<tr><td>voucherInfoBalanceValidityOffset</td><td>WalletInfoRequest</td><td>A relative offset from the current date when a given balance, charged with a voucher, becomes valid.</td></tr>
<tr><td>voucherInfoBalanceValidityStart</td><td>WalletInfoRequest</td><td>A fixed date in the future when a given balance, charged with a voucher, becomes valid.</td></tr>
<tr><td>voucherInfoBalanceValidityType</td><td>WalletInfoRequest</td><td>The units of the relative offset from the current date when the balance becomes valid.
voucherInfoBalanceValidityType returns the following values:
<table>
<tr><th>Value</th><th>Meaning</th></tr>
<tr><td>0</td><td>Hours</td></tr>
<tr><td>1</td><td>Months</td></tr>
</table></td></tr>
<tr><td>voucherInfoMissingBalancePolicy</td><td>WalletInfoRequest</td><td>Indicates what to do if the specified balance type is missing from the list of existing balances for the voucher.
voucherInfoMissingBalancePolicy returns the following values:
<table>
<tr><th>Value</th><th>Name</th><th>Meaning</th></tr>
<tr><td>0</td><td>allow</td><td>Create the balance (and bucket, if applicable) and set it to the specified value</td></tr>
<tr><td>1</td><td>fail</td><td>Reject the recharge (Invalid Recharge Value)</td></tr>
<tr><td>2</td><td>ignoreBalance</td><td>Skip this balance and process the rest of the recharge</td></tr>
</table></td></tr>
<tr><td>voucherInfoNewBucket</td><td>WalletInfoRequest</td><td>If this value is set to true, the voucher value will be added to the balance as a new bucket.</td></tr>
<tr><td>voucherInfoReplaceBalance</td><td>WalletInfoRequest</td><td>If this value is set to true, all existing buckets of the balance will removed, and a new bucket is created with the specified voucher value.</td></tr>
</table>

<table>
<tr><th>Concept Label</th><th>Available</th><th>Comments</th></tr>
<tr><td>voucherInfoValue</td><td>WalletInfoRequest</td><td>The voucher balance recharge details.</td></tr>
<tr><td>voucherInfoVoucher</td><td>WalletInfoRequest</td><td>The database key of the voucher being redeemed.</td></tr>
<tr><td>voucherInfoVoucherNumber</td><td>WalletInfoRequest</td><td>The voucher number of the voucher being redeemed.</td></tr>
<tr><td>voucherInfoVoucherSerialNumber</td><td>WalletRechargeRequest</td><td>Populates the Voucher Serial Number in a DCD AVP, so that it may be used to audit and track the voucher redemption.</td></tr>
<tr><td>voucherInfoWalletExpiryExtension</td><td>WalletInfoRequest</td><td>The extension period to apply to the wallet expiry date of the recharged wallet.</td></tr>
<tr><td>voucherInfoWalletExpiryExtensionPolicy</td><td>WalletInfoRequest</td><td>Indicates how to apply the wallet expiry extension period to the wallet expiry date.</td></tr>
<tr><td>voucherInfoWalletExpiryExtensionType</td><td>WalletInfoRequest</td><td>The unit of the expiry extension for the wallet that the voucher will recharge (example: hours or months).
voucherInfoWalletExpiryExtensionType returns the following values:
<table>
<tr><th>Value</th><th>Meaning</th></tr>
<tr><td>0</td><td>Hours</td></tr>
<tr><td>1</td><td>Months</td></tr>
</table>
</td></tr>
<tr><td>voucherRechargeFailureDateTime</td><td>WalletRechargeRequest</td><td>Returns the timestamp of any previous voucher recharge failure. If there has not been a previous voucher recharge failure, then zero (0) is returned.</td></tr>
<tr><td>voucherRechargeFailureFlag</td><td>WalletRechargeRequest</td><td>Returns the value of one (1) if the voucher is not redeemed and a failed voucher redeem attempt has been made.
Returns zero (0) for all other voucher states. For example, if a redeem attempt has never been made for the voucher or if the voucher has been redeemed successfully.</td></tr>
<tr><td>voucherTypeName</td><td>WalletInfoRequest</td><td>Returns the name of the type of voucher being redeemed.
Note: Voucher type name is only available if a positive value is defined for <code>voucherTypeCacheSize</code> in the ccsActions section. When this cache is configured, CCS will be able to use the batch of the voucher being redeemed to lookup the voucher type, and that name will then be available to the ccsConcept voucherTypeName.</td></tr>
</table>

Chapter 2

Split Charging and Voucher Domains

Overview

Introduction

This chapter explains how the DCD components handle split charging and voucher redemption when wallets and vouchers are hosted by different billing domains.

In this chapter

This chapter contains the following topics.

Wallets and Vouchers Split Recharging

Introduction

CCS supports charging services for redeeming vouchers and updating wallets when they reside on the same billing domain. The DCD product provides functionality that allows the Prepaid Charging product to extend this support to separate voucher and wallet domains in CCS.

DCD can be configured to support a CCS-based convergent billing solution where separate billing systems are used to host:

- Vouchers and redemption functions
  Example: Oracle VWS-Voucher Management
- Wallets and charging functions
  Example: A third-party billing system.

In order to achieve this, DCD can be extended using ccsConcepts required to support voucher redemption.

Key Components

The key components that enable this split-domain architecture with DCD are as follows:

Component	Description	Further Information
ACS voucherDelegator	Configures the diamActions which support different billing domains for recharges.	*voucherDelegator* (on page 16)
ccsConcepts	Support voucher redemption variables.	Voucher details

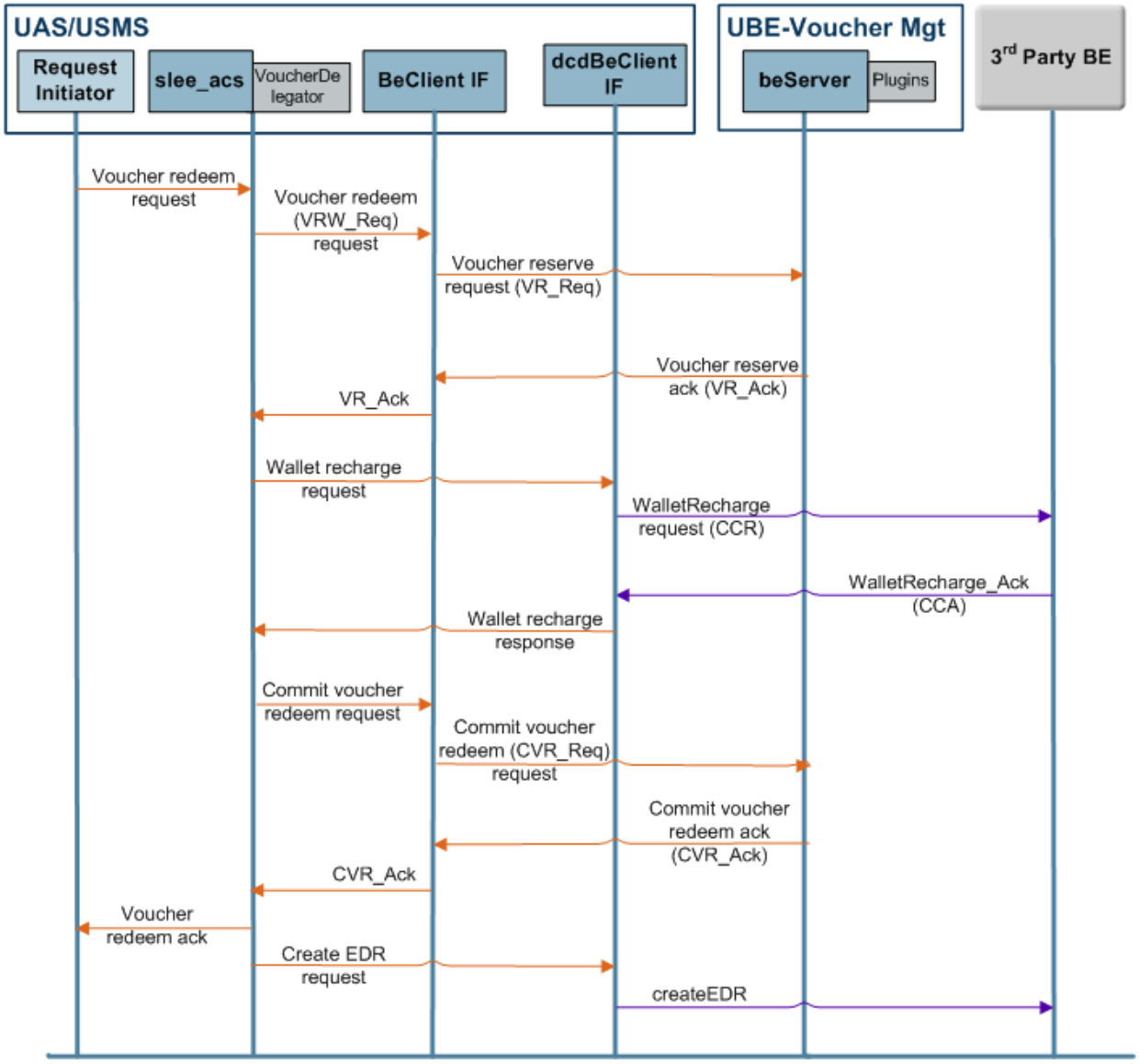
voucherDelegator

The voucherDelegator is a slee_acs process which implements the following diamActions enabling split domain recharging over DCD:

Action	Expected Response	Description
BadPIN	• CCR named `BadPINRequest`, and • CCA named `BadPINResponse`	Sends a CCR to the BE, notifying that a given MSISDN has failed to redeem a voucher.
CreateEDR	• CCR named `CreateEDRRequest`, and • CCA named `CreateEDRResponse`	Sends a list of tags and values as AVPs to the third-party BE which will be added to the BE EDR.
WalletRecharge	• CCR named `WalletRechargeRequest`, and • CCA named `WalletRechargeResponse`	Sends a CCR to the third-party BE with a wallet rechage request, and expects a CCA with a wallet recharge response.

Diagram

Here is an example of the split-domain wallet and voucher recharging process. Note that the third-party billing engine server is also the Diameter Server in this example.



Split Recharging Process

This table describes the stages involved in redeeming a voucher using VWS-Voucher Management and recharging a wallet on a third-party domain.

Stage	Description
1	Voucher redemption is triggered using any of the following methods: • IVR feature nodes in a control plan • Interaction with a customer services representative (who uses the Voucher Management screen) • (If MM is installed) Short Messages sent from the subscriber's handset, and • (If USSD GW is installed) menus and fast access.

Stage	Description
	The information from the request initiator is passed to the voucherDelegator, which sends a message to the relevant BeClientIF process to reserve the voucher.
2	The BeClientIF sends a Voucher Reserve (VR_Req) request to VWS-Voucher Management.
3	VWS-Voucher Management checks whether: • This VWS holds the details for the requested voucher • The voucher PIN number is correct • If the voucher can be redeemed If the voucher can be redeemed, VWS-Voucher Management reserves the voucher and passes a Voucher Reserve acknowledgment (VR_Ack) back to the voucherDelegator.
4	The voucherDelegator processes the message and sends a Wallet Recharge request to the dcdBeClientIF in an attempt to recharge the wallet on the Diameter Server (in this case the third-party BE).
5	The dcdBeClientIF constructs a CCR with a WalletRecharge action and interrogates the Diameter Server for wallet recharge.
6	The Diameter Server checks whether: • The details for the requested wallet • Whether the wallet state allows it to be updated If the wallet can be recharged, the Diameter Server sends a CCA response back to the dcdBeClientIF with a Wallet Recharge acknowledgment, which is reported back to the voucherDelegator.
7	The voucherDelegator then sends a Commit Voucher Redeem (CVR_Req) request to the BeClientIF which is sent to VWS-Voucher Management for redeeming the voucher. If the voucher redeem succeeds, VWS-Voucher Management responds with a Commit Voucher Redeem acknowledgement (CVR_Ack) to the BeClientIF which is reported back to the voucherDelegator.
8	The voucherDelegator processes the message and informs the request initiator of the successful voucher redemption wallet recharge.
9	The voucherDelegator then initiates the createEDR action for the relevant EDRs to be produced on the Diameter Server.
10	If either the voucher redemption or the wallet recharge failed, appropriate Not Acknowledgment (Nack) messages at each stage and this is recorded in the final set of EDRs generated.

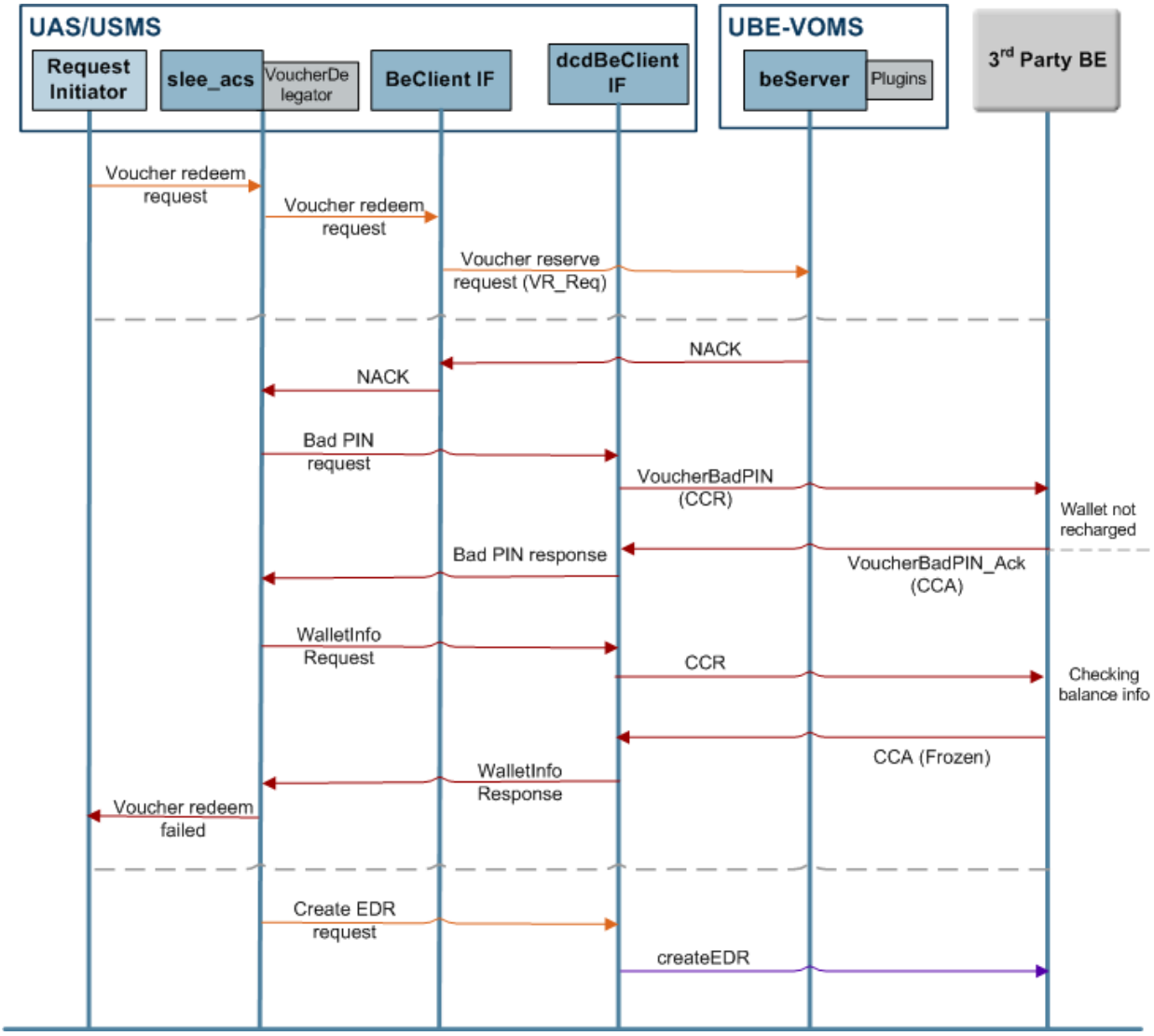
Bad PIN

Description

The ACS voucherDelegator also handles BadPIN processing and notifies the BE that controls wallet recharging if a subscriber (MSISDN) has failed to redeem a voucher.

Diagram

Here is an example of Bad PIN processing on a split-domain wallet and voucher network. Note that the third-party billing engine server is also the Diameter Server in this example.



Checking Voucher Bad PIN

This table describes an unsuccessful voucher redemption due to a Bad PIN.

Stage	Description
1	When VWS-Voucher Management receives a Voucher Reserve (VR_Req) request, it checks whether: • The VWS holds the details for the requested voucher • The voucher PIN number is correct • If the voucher can be redeemed If the voucher PIN is incorrect, VWS-Voucher Management updates the bad PIN counter and returns a Bad PIN Increase (BPIN) count to the BeClientIF which informs the voucherDelegator.

Stage	Description
2	On successive Bad PIN attempts, the voucherDelegator cancels the voucher redemption and sends a Revoke Voucher Redeem (RVR) request to the BeClientIF which is passed onto VWS-Voucher Management.
3	VWS-Voucher Management responds with a Revoke Voucher Redeem (RVR_Ack) acknowledgement which means that the voucher redemption request stands cancelled.
4	The voucherDelegator then sends a Bad PIN request to the dcdBeClientIF in an attempt to cease any transactions on the wallet domain.
5	The dcdBeClientIF constructs a CCR with a BadPINRequest action and notifies the Diameter Server.
6	The Diameter Server confirms and sends a CCA back to the dcdBeClientIF with a BadPINResponse, which is reported to the voucherDelegator.
7	The voucherDelegator processes the message and informs the request initiator that the voucher redemption was unsuccessful.
8	The voucherDelegator then initiates the createEDR action for the relevant EDRs to be produced on the Diameter Server.

Chapter 3

SCAP Compliance

Overview

Introduction

This chapter explains the summary of changes that affect DCD to make it SCAP compliant.

In this chapter

This chapter contains the following topics.

SCAP

Introduction

Ericsson SCAP is a vendor specific protocol that utilizes the Draft version 8 of the Diameter Base Protocol (the predecessor to *RFC 3588)*.

The NCC Diameter protocol is based upon the final version of *RFC 3588*. The base protocol is further extended to support the use of CCR (Credit-Control-Request) / CCA (Credit-Control-Answer) messages, described by *RFC 4006*.

SCAP, on the other hand, adds Attribute Value Pair (AVP)s to Diameter base protocol's Accounting-Request (ACR) and Accounting-Answer (ACA) commands.

DCD may be reconfigured to permit it to interact with an Ericsson SCAP compliant server as a SCAP client. Variances between the SCAP and normal Credit-Control charging approaches available within DCD are described below.

Note: The necessary changes will take effect only if the *enableDraft8* (on page 35) is set to `true`.

Application-Identifiers Values

This table lists the different Application-identifier values between the Diameter Base Protocol Draft 8 and RFC 3588 final version.

Draft 8	Final
-	Diameter Common Messages 0
NASREQ 1	NASREQ 1
CMS Security 2	Mobile-IP 2
Mobile IP 4	-
Relay 0xffffffff	Relay 0xffffffff

Message Header

This table describes the changes to message header values.

Section Heading	Comment
Vendor-Id	This should be changed in outbound messages (at the Oracle Diameter stack level). See *Vendor-Id* (on page 40) for SCAP specific changes.
T-flag	The T-flag (retransmit) is used by a Diameter client to indicate to the server that a message has been retransmitted (possibly due to loss of message). This flag is not supported by Draft 8, and hence it is not set while retransmitting duplicate DCD messages to SCAP.

Supported AVPs

This table describes the changes to the supported AVPs to enable SCAP compliance.

Supported AVP	Comment
Event-Timestamp	DCD to send vendor specific version for Ericsson SCAP.
Inband-Security-Id	This AVP is not sent during capabilities exchange, if SCAP support is configured, that is, when *enableDraft8* (on page 35) is set to `true`.

Result-Codes

This table describes the changes to the Result-Codes to enable SCAP compliance.

Result Code	[DRAFT8] Value	[3588] Value
DIAMETER_UNSUPPORTED_TRANSFORM	5010	-
DIAMETER_NO_COMMON_APPLICATION	5011	5010
DIAMETER_UNSUPPORT_VERSION	5012	5011
DIAMETER_UNABLE_TO_COMPLY	5013	5012
INVALID_BIT_IN_HEADER	5014	5013
INVALID_AVP_LENGTH	5015	5014
INVALID_MESSAGE_LENGTH	5016	5015
INVALID_AVP_BIT_COMBO	5017	5016
DIAMETER_NO_COMMON_SECURITY	-	5017

Chapter 4

Configuration

Overview

Introduction

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

In this chapter

This chapter contains the following topics.

Configuration Overview

Introduction

This topic provides a high level overview of how the DCD 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 Charging Driver is configured by the following components:

Component	Locations	Description	Further Information
eserv.config	all SLC machines	DCD is configured by the Diameter section of **eserv.config**.	*eserv.config Configuration* (on page 24)
eserv.config	all SLC machines	The WalletInformation cache is configured in the CCS section of **eserv.config**.	*CCS eserv.config Configuration* (on page 25)
SLEE.cfg	all SLC machines	The SLEE interface is configured to include the DCD service.	*SLEE.cfg Configuration* (on page 26) and the *SLEE Technical Guide*

Component	Locations	Description	Further Information
acs.conf	all SLC machines	Configures the diamActions library.	*acs.conf Configuration* (on page 32)
oracleConfig.xsd	SMS	Defines acceptable structure for XML.	
oracleConfigWorking.xml	SMS	The editable configuration.	Configuration Management
oracleConfigMaster.xml	SMS	The deployed (live) configuration used to generate the **eserv.config**.	Configuration Management

Note: The **.xsd** and **.xml** files are present only when the Configuration Management editor is installed.

eserv.config Configuration

Introduction

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

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

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

Configuration File Format

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

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

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

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

or

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

or

```
    { name="route6"
```

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

eserv.config Files Delivered

Most applications come with an example **eserv.config** file named **eserv.config.example**. The example file for DCD is:

```
/IN/service_packages/DCD/etc/eserv.config.dcd.example
```

Editing the File

You can edit the **eserv.config** file by using one of the following:

- The Configuration Management editor
- A text editor

Warning: If you are using the Configuration Management editor, you cannot use a text editor to edit the DCD section.

Text Editor

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

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

Loading eserv.config Changes

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

Diameter eserv.config Configuration

The **eserv.config** file must be configured to enable the DCD to work. Most of the necessary DCD configuration in **eserv.config** is done at installation time by the configuration script. However, realms and hosts need to be configured.

Note: The DCD configuration options in **eserv.config** are explained in the section on the *diameterBeClient background process* (on page 79).

CCS eserv.config Configuration

Introduction

DCD maintains a WalletInformation cache. To enable updates to the WalletInformation cache, the service handles for the services using DCD must be mapped to the `libdcdCcsSvcExtra.so` library. You will need to configure the mapping in the `CCS.ccsServiceLibaray.ccsPluginExtend` of **eserv.config**.

ccsServiceLibrary Configuration

Here is an example of the ccsServiceLibrary configuration for DCD in the CCS section of **eserv.config**.

```
ccsServiceLibrary = {
    ccsPluginExtend = [
        {
            library="libdcdCcsSvcExtra.so"
        }
    ]
}
```

RAR Configuration

You enable processing of re-authorization requests (RARs) in Diameter Control Agent (DCA).

For information about enabling RAR processing in DCA, see the RAR configuration section in *Diameter Control Agent Technical Guide.*

When RAR processing is enabled, DCA marks the first INITIAL_REQUEST as RAR enabled to allow DCD to process any subsequent RARs. You can specify the amount of time DCD should wait for a response to an RAR sent to the Diameter client via DCA by configuring the `rarSleeTimeout` parameter in the DIAMETER section of the **eserv.config** file:

```
DIAMETER = {

    DCD = {
        rarSleeTimeout = seconds

    }
}
```

where *seconds* is the amount of time in seconds that the DCD will wait for a response to an RAR sent to the Diameter client via DCA.

After this timeout lapses, DCD responds to the Diameter server with a re-authorization acknowledgement (RAA) containing the DIAMETER_UNABLE_TO_DELIVER (3002) result code. If the DCA responds after the DCD has sent an RAA, the DCA response is ignored.

Note: The `rarSleeTimeout` value must be greater than 0 (zero), and is set to 10 by default. You are recommended to set the value of the `rarSleeTimeout` greater than the value of the `rarClientTimeout` in the DCAInstances section of **eserv.config**.

To disable the timeout, set `rarSleeTimeout` to 0 (zero). A value of 0 (zero) means that no SLEE timeout will be used.

Note: If RAR processing is not enabled in DCA, then DCD will respond to the diameter server with an RAA containing the corresponding result code set in the `rarResultCode`.

SLEE.cfg Configuration

Introduction

The **SLEE.cfg** file must be configured to enable the DCD to work. All necessary SLEE configuration is done at installation time by the configuration script.

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

See *SLEE Technical Guide* for details about SLEE configuration.

DCD SLEE Configuration

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

```
INTERFACE=dcdBeClient diameterBeClient.sh /IN/service_packages/DCD/bin EVENT
```

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

PeerSchemes Configuration Section

Example PeerSchemes

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

Note: The `PeersSchemes` or `Peers` section is mandatory.

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

    Peers = [

    {
        name = "host1"

        scheme = [ "scheme1", "schmeme2" ]

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

        peer_group = "host1"

        transport = "tcp"

        initiation = "connect"

        RemoteAddresses = [
            "192.168.1.10"
        ]

        remote_port = 3868

        netmask6Bits = 128

        netmaskBits = 32

        permittedInstances = 0

        reqSctpInboundStreams = 8
        reqSctpOutboundStreams = 8

        sctp_hbinterval = 1000

        watchdogPeriod = 30

        connectionTimeout = 30

        inBufferSize = 0
        outBufferSize = 0
```

```
        } # end of Peer host1

        {
            Peer_Host2_Parameters
        }
        ]
    } # End of Scheme A

    {
        schemeName = "SchemeB"

        Peers = [

            Parameters_for_SchemeB_peers

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

`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 found within the `Peers` section.

Note: The `PeerSchemes` or `Peers` section is mandatory.

The available parameters are:

```
        {
            name = "host1"

            scheme = [ "scheme1", "schmeme2" ]

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

            peer_group = "host1"

            transport = "tcp"

            initiation = "connect"

            RemoteAddresses = [
                "192.168.1.10"
            ]

            remote_port = 3868

            netmask6Bits = 128

            netmaskBits = 32

            permittedInstances = 0
```

```
            reqSctpInboundStreams = 8
            reqSctpOutboundStreams = 8

            sctp_hbinterval = 1000

            watchdogPeriod = 30

            connectionTimeout = 30

            inBufferSize = 0
            outBufferSize = 0

        } # end of Peer host1
```

Note: All the peer configuration items from "transport" on down can also have global defaults set in the `DIAMETER.DCD` section. See *DCD* (on page 33).

connectionTimeout

Syntax:	`connectionTimeout = timeout`
Description:	The timeout for re-establishing connections (*RFC 3588* Tc).
Type:	Integer
Optionality:	Optional
Allowed:	In seconds
Default:	30
Example:	`connectionTimeout = 30`

inBufferSize

Syntax:	`inBufferSize = size`
Description:	The size of the socket receive buffer.
Type:	Integer
Optionality:	Mandatory
Allowed:	in bytes
Default:	0 (means to use the OS default)
Example:	`inBufferSize = 0`

initiation

Syntax:	`initiation = "action"`
Description:	How to start the connection.
Type:	String
Optionality:	Mandatory
Allowed:	• listen = listen for incoming connections • connect = connect an outgoing connection.
Example:	`initiation = "connect"`

name

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

Example: `name = "host1"`

netmaskBits

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

netmask6Bits

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

outBufferSize

Syntax: `outBufferSize = size`
Description: The size of the socket send buffer.
Type: Integer
Optionality: Mandatory
Allowed: in bytes
Default: 0 (means to use the OS default)
Example: `outBufferSize = 0`

peer_group

Syntax: `peer_group = "name"`
Description: The peer group that the peer host connects to. If multiple peer hosts belong to the peer group, only one peer host is connected.

The peer group works in failover mode only; it does not support round-robin mode.
Type: String
Optionality: Optional
Default: Defaults to the value specified by the `name` parameter. For example, if the `name` parameter is set to "host1", `peer_group` defaults to "host1".
Example: `peer_group = "host1"`

permittedInstances

Syntax: `permittedInstances = number`
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 that will be checked against the OriginHost AVP during the capabilities exchange.

Type: String

Optionality: Mandatory

Notes: This parameter accepts at least one host and will accept any one of them when it gets the Capabilities Exchange Answer. These are the hosts allowed to talk to this client.

Example:
```
permittedOriginHosts = [
"host1.realm1.oracle.com"
"host2.realm1.oracle.com"
]
```

remote_port

Syntax: `remote_port = number`

Description: The remote post number.

Type: Integer

Optionality: Optional

Default: Defaults to the RFC specified 3868

Example: `remote_port = 3868`

RemoteAddresses

Syntax: `remoteAddresses = ["ipaddress"]`

Description: The list of remote IP addresses.

Type: Array of string parameters

Optionality: Mandatory

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

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

reqSctpInboundStreams

Syntax: `reqSctpInboundStreams = number`

Description: The number of requested inbound sctp streams.

Type: Integer

Optionality: Mandatory

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

Example: `reqSctpInboundStreams = 8`

reqSctpOutboundStreams

Syntax: `reqSctpOutboundStreams = number`

Description: The number of requested outbound sctp streams.

Type: Integer

Optionality: Mandatory

Notes: There is no guarantee you will actually get these.
Example: `reqSctpOutboundStreams = 8`

`scheme`

Syntax: `scheme = [ "scheme", "scheme" ]`
Description: The list of schemes to which you have limited the peer.
Type: Array of strings
Optionality: If using `PeerSchemes` this is mandatory.
Allowed: Names of configured schemes.
Example: `scheme = [ "scheme1", "scheme2" ]`

`sctp_hbinterval`

Syntax: `sctp_hbinterval = interval`
Description: The interval for sctp heartbeats.
Type: Integer
Optionality: Optional
Allowed: in milliseconds
Default: 1000
Example: `sctp_hbinterval = 1000`

`transport`

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

Default: If not specified, then it uses the one from the global DCD section.
Example: `transport = "tcp"`

`watchdogPeriod`

Syntax: `watchdogPeriod = period`
Description: The quiet period before sending a DWR. (*RFC 3588* Tw).
Type: Integer
Optionality: Mandatory
Allowed: in seconds
Default: 30
Example: `watchdogPeriod = 30`

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.

DCD acs.conf Configuration

On installation, the following line is added to the **acs.conf**.

```
ChassisPlugin diamActions.so
```

DCD

Introduction

The `DCD` section holds global configuration for the DCD application.

Note: This section is optional.

DCD Parameters

Here are the parameters for the `DCD` section.

```
DCD = {
    serviceDomainInterfaceName = "dcdBeClient"
    loggedNotificationPeriod = 300
    loggedInvalidPeriod = 300

    databaseCacheValiditySeconds = 18000

    Origin-Host = "ocpc.oracle.com"

    Origin-Realm = "ocpc.oracle.com"

    serviceContextID = "ocpc@oracle.com"

    Auth-Application-Id = 4

    Vendor-Specific-Application-Id = [
        {
        Vendor-Id = 193
        Auth-Application-Id = 4
        Acct-Application-Id = 19302
        }
    ]

    enableDraft8 = false

    # scheme = "SchemeA"

    honour_disconnect = true

    Vendor-Id = 16247

    Product-Name = ""

    timeout_x = 30000000

    transmit_limit = 2

    trace_debug_flags = "all"
```

```
        statistics_interval = 300

        statsCollection = false

        NotificationFilter = {
            CollectiveNotice = true
            PerPeerNotice = {
                initial_request = true
                update_request = true
                termination_request = true
                event_request = true
                unknown_request = true
                total_request = true
                request_timeout = true
                non_request_type_result_codes = true
            }

        }

    }
```

You may also set defaults for peer connection settings here. See *Peer Host Parameters* (on page 28) for details of the available options.

`Auth-Application-Id`

Syntax:	`Auth-Application-Id = ID`
Description:	This AVP value is set to the Credit-Control as DCD uses Credit-Control capability by default.
Type:	Integer
Optionality:	Optional (not sent if not set).
Allowed:	
Default:	Not sent
Notes:	• This AVP can be configured as a sub-AVP to support *Vendor-Specific-Application-Id* (on page 40). • In case of vendor specific configuration, this value should NOT be set globally.
Example:	`Auth-Application-Id = 4`

`Acct-Application-Id`

Syntax:	`Acct-Application-Id = ID`
Description:	This AVP value is configured as a sub-AVP to support *Vendor-Specific-Application-Id* (on page 40).
Type:	Integer
Optionality:	Optional (not sent if not set).
Allowed:	
Default:	Not sent
Notes:	Either one, or both of `Auth-Application-Id` or `Acct-Application-Id` should be set in order to support *Vendor-Specific-Application-Id* (on page 40).
Example:	`Acct-Application-Id = 12300`

`diameterMessageLength`

Syntax:	`diameterMessageLength = size`
Description:	Maximum size of CCA packet received.

Type: Integer
Optionality: Optional (default used if not set)
Allowed:
Default: 8192
Notes:
Example: `diameterMessageLength = 32768`

checkDefinedAvpFlags

Syntax: `checkDefinedAvpFlags = true|false`
Description: Whether to check incoming AVPs for flags that are defined in the base Diameter protocol, or in any of the Diameter application specifications, so that unknown mandatory AVPs can be excluded.
Type: Boolean
Optionality: Optional (default used if not set)
Allowed: true – Check flags of incoming AVPs.
false – Do not check flags of incoming AVPs.
Default: true
Example: `checkDefinedAvpFlags = false`

databaseCacheValiditySeconds

Syntax: `databaseCacheValiditySeconds = seconds`
Description: Defines how long to keep data from the database after loading it.
Type: Integer
Optionality: Optional
Allowed: Seconds
Default: 18000 seconds (5 hours)
Example: `databaseCacheValiditySeconds = 18000`

enableDraft8

Syntax: `enableDraft8 = true|false`
Description: Indicates if Draft 8 version of the Diameter base protocol should be used. This is required to support the `enableScap` (on page 43) parameter.
Type: Boolean
Optionality: Optional (default used if not set).
Allowed: true, false
Default: `false`
Notes:
Example: `enableDraft8 = false`

enableDraft8

Syntax: `excludeWhenEmpty = true|false`
Description: Causes DCD to not send a parent Attribute-Value Pair (AVP) if all its child AVPs are empty.

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

honour_disconnect

Syntax:	`honour_disconnect = true|false`
Description:	This is whether to use the RFC 3588 disconnect logic.
Type:	Boolean
Optionality:	Optional
Allowed:	true, false
Default:	false
Notes:	By default the RFC3588 disconnect logic is enabled. If your server incorrectly sends DPR messages, then disable this by setting to false.
Example:	`honour_disconnect = true`

loggedInvalidPeriod

Syntax:	`loggedInvalidPeriod = period`
Description:	This is how often to announce the number of invalid messages.
Type:	Integer
Optionality:	Optional
Allowed:	Seconds
Default:	300
Example:	`loggedInvalidPeriod = 300`

loggedNotificationPeriod

Syntax:	`loggedNotificationPeriod = period`
Description:	This is how often to announce the number of recent message parse errors.
Type:	Integer
Optionality:	Optional
Allowed:	Seconds
Default:	300
Example:	`loggedNotificationPeriod = 300`

NotificationFilter

Syntax:	`NotificationFilter = {`
	`CollectiveNotice =`
	`PerPeerNotice = {}`
	`}`

Description:	This section sets flags allowing the appropriate notification to be enabled or suppressed.
Type:	Section of boolean parameters
Optionality:	Optional, default used if not set
Default:	All values default to true (that is, statistics recording in DCD alarm log enabled).

Notes: Settings are only applicable to statistics that are recorded in the DCD alarm log.

Example:

```
NotificationFilter = {

        CollectiveNotice = true
        PerPeerNotice = {
            initial_request = true
            update_request = true
            termination_request = true
            event_request = true
            unknown_request = true
            total_request = true
            request_timeout = true
            non_request_type_result_codes = true
        }
    }
```

CollectiveNotice

Syntax: `collectiveNotice = true|false`

Description: Enables or suppresses the recording of CCR request type statistics.

Type: Boolean

Optionality: Optional (default used if not set).

Allowed:

- true – Enable CCR request type statistics
- false – Suppress CCR request type statistics

Default: true

Notes: Counts are collective across all peers and not separated on a per peer basis.

Example: `collectiveNotice = true`

PerPeerNotice

Syntax: `PerPeerNotice = {`*request_type* `= true|false> list}`

Description: Enables/suppresses statistic counts per peer for both CCR and CCA for the listed CC_Request_types.

Type: Section of boolean parameters

Optionality: Optional (default used if not set).

Allowed: Each request type can be set to:

- true (statistic enabled)
- false (statistic suppressed)

See example for list of request types.

Default: All values default to true (statistic enabled).

Notes:

Example:

```
PerPeerNotice = {
    initial_request = true
    update_request = true
    termination_request = true
    event_request = true
    unknown_request = true
    total_request = true
    request_timeout = true
    non_request_type_result_codes = true
}
```

Origin-Host

Syntax: `Origin-Host = "diameterId"`
Description: The Diameter AVP Origin-Host - fully qualified domain name.
Type: String
Optionality: Optional
Notes: May be specified for each host. See *HostSpecificData Parameters* (on page 73).
Example: `Origin-Host = "ocpc.oracle.com"`

Origin-Realm

Syntax: `Origin-Realm = "diameterId"`
Description: The Diameter AVP Origin-Realm.
Type: String
Optionality: Optional
Notes: May be specified for each host. See *HostSpecificData Parameters* (on page 73).
Example: `Origin-Realm = "ocpc.oracle.com"`

Product-Name

Syntax: `Product-Name = "name"`
Description: The Product name for CER/CEA.
Type: String
Optionality: Optional
Default: empty
Example: `Product-Name = ""`

rarResultCode

Syntax: `rarResultCode = integer`
Description: What happens when diameterBeClient receives a Re-Auth-Request
Type: Integer
Optionality: Optional (default used if not set)
Allowed: Any integer
Default: `rarResultCode = 3001, DIAMETER_COMMAND_UNSUPPORTED`
Notes: When `diameterBeClient` receives a RAR:

- If the value is omitted or specified as 3001, it logs an error message and responds with Re-Auth-Answer (Result-Code=3001).
- If a value other than 3001 is specified, it does not log an error message and responds with Re-Auth-Answer (Result-Code=the specified `rarResultCode`).

diameterBeClient takes no further action and does not send the Re-Auth-Request to slee_acs.
Example: `rarResultCode = 3001`

rarsleeTimeout

Syntax: `rarSLEETimeout = int`
Description: The number of seconds DCD will wait for a response from a RAR sent to the Diameter client via DCA
Type: Integer
Optionality: Optional (default used if not set)

Allowed: >=0
Default: 10

scheme

Syntax: `scheme= "type"`
Description: The global scheme to use.
Type: String
Optionality: If you use the `peerSchemes` section, then this is mandatory.
Example: `scheme = "schemeA"`

serviceDomainInterfaceName

Syntax: `serviceDomainInterfaceName = "name"`
Description: The interface name of the Diameter BE client (in **SLEE.cfg**).
Type: String
Optionality: Optional
Default: dcdBeClient
Example: `serviceDomainInterfaceName = "dcdBeClient"`

serviceContextID

Syntax: `serviceContextID = "ContextID"`
Description: The Diameter AVP Service-Context-Id.
Type: String
Optionality: Mandatory
Example: `serviceContextID = "ocpc@oracle.com"`

statistics_interval

Syntax: `statistics_interval = seconds`
Description: This is how often to record aggregate (average, min, max) latency statistics.
Type: Integer
Optionality: Mandatory
Allowed: In seconds.
Default: 300 (that is, 5 minutes)
Example: `statistics_interval = 300`

statsCollection

Syntax: `statsCollection = true|false`
Description: Whether to collect statistics in DCD.
Type: Boolean
Optionality: Optional (default used if not set)
Allowed: true – Enable statistics collection.
false – Disable statistics collection.
Default: true
Example: `statsCollection = false`

timeout_x

Syntax: `timeout_x = timer`
Description: The RFC 4006 Tx timer.
Type: Integer
Optionality: Optional
Allowed: In microseconds
Default: 30000000
Example:
```
timeout_x = 30000000
```

trace_debug_flags

Syntax: `trace_debug_flags = "flags"`
Description: The debug flags to turn on if tracing (in the client) is requested by the actions library.
Type: String
Optionality: Optional
Default: "all"
Example:
```
trace_debug_flags = "all"
```

transmit_limit

Syntax: `transmit_limit = limit`
Description: The maximum number of retransmits (including the original transmission) allowed for a message.
Type: Integer
Optionality: Optional
Example:
```
transmit_limit = 2
```

Vendor-Id

Syntax: `Vendor-Id = ID`
Description: The Vendor ID for CER/CEA.
Type: Integer
Optionality: Optional
Default: 16247 (the Oracle Vendor-Id)
Notes:
- This AVP can be configured as a sub-AVP to support *Vendor-Specific-Application-Id* (on page 40).
- It is mandatory to specify `vendorId` when configuring vendor specific AVPs for SCAP.

Example:
```
Vendor-Id = 16247
```

Vendor-Specific-Application-Id

Syntax:
```
Vendor-Specific-Application-Id = [
    {
    Vendor-Id = int
    #Auth-Application-Id = int
    Acct-Application-Id = int
    }
]
```
Description: Lists the Vendor specific AVPs required for enabling SCAP.
Type: Parameter array

Optionality:	Optional (default used if not set).
Allowed:	
Default:	Parameter array is not specified.
Notes:	Do NOT set the global *Auth-Application-Id* (on page 34) if configuring this parameter array.

Example:	`Vendor-Specific-Application-Id = [`
	`{`
	`Vendor-Id = 123`
	`Acct-Application-Id = 12345`
	`}`
	`]`

DomainTypes

Introduction

The `DomainTypes` section lists all DCD domain types and the associated configuration.

Note: This section is mandatory, and it must include one domain type, with a name and scheme defined.

DomainTypes Parameters

Here is an example high-level structure showing the parameters for the `DomainTypes` section.

```
DomainTypes = [
{
    name = "DIAMETER"

    schemeName = "SchemeA"

    routing = "Round Robin"

    voidUnusedReservation = false

    releaseOnLowCredit = false

    defaultSessionFailover = 0
    defaultEventFailover = 0
    defaultFailureHandling = 0

    balanceEnquiryMethod = "balanceCheck"

    includeDcdCdrFields = false

    defaultFixedCostDuration = 86400

    conversionScale = 1

    enableScap = false
    overwriteZeroCallAnswerTime = false

    Domains = [
        {First_Domain
        }

        {Next_Domain
        }
```

```
        ]

        AVPs = [

            {First_AVP
            }

            {Next_AVP
            }

            {...
            }

        ]

    }
```

balanceEnquiryMethod

Syntax:	`balanceEnquiryMethod = "method"`
Description:	The method to use to allow balance queries.
Type:	String
Optionality:	Optional (default used if not set)
Allowed:	• "balanceCheck" uses a Balance Check message with a Service Identifier set to "Information" to trigger the query, or • "reqActionViewBalance" uses a special Requested-Action AVP with a value of VIEW_BALANCE(5) to trigger the query
Default:	"balanceCheck"
Notes:	
Example:	`balanceEnquiryMethod = "balanceCheck"`

defaultEventFailover

Syntax:	`defaultEventFailover = failover`
Description:	The default event failover.
Type:	Integer
Optionality:	Optional (default used if not set).
Allowed:	The failover corresponds to the values defined for Credit Control Failure Handling AVP in *RFC 4006*: • 0 – Terminate • 1 – Continue • 2 – Retry and Terminate
Default:	0 (Failover not supported)
Notes:	Until the BE responds for the event, the failover behavior is determined by this and the `defaultSessionFailover` parameter.
Example:	`defaultEventFailover = 0`

defaultFailureHandling

Syntax:	`defaultFailureHandling = number`
Description:	How to behave until a Credit-Control-Failure-Handling AVP (as defined in *RFC 4006*) is received from the server.
Type:	Integer
Optionality:	Optional (default used if not set).

Allowed:
- 0 – Terminate
- 1 – Continue
- 2 – Retry and Terminate

Default: 0 (Terminate)

Notes:

Example: `defaultFailureHandling = 0`

defaultFixedCostDuration

Syntax: `defaultFixedCostDuration = duration`

Description: The default time to use for free and fixed cost calls.

Type: Integer

Optionality: Optional (default used if not set).

Allowed: In seconds. May be any positive value.

Default: 86400 (one day)

Notes:

Example: `defaultFixedCostDuration = 86400`

defaultSessionFailover

Syntax: `defaultSessionFailover = failover`

Description: The default session failover.

Type: Integer

Optionality: Optional (default used if not set).

Allowed: Corresponds to the values defined for Credit Control Failure Handling AVP in *RFC 4006*:
- 0 – Terminate
- 1 – Continue
- 2 – Retry and Terminate

Default: 0 (Failover not supported)

Notes: Until the BE responds in a session, the failover behavior for a CC session is determined by this.

Example: `defaultSessionFailover = 0`

enableScap

Syntax: `enableScap = true|false`

Description: If set to true, support for Ericsson SCAP (Service Charging Application Protocol) to DCD is enabled. This ensures SCAP type accounting messages are used for credit control, rather than CCR/CCA.

Type: Boolean

Optionality: Optional (default used if not set).

Allowed:
- true – Enabled for SCAP compliance
- false – Disabled for SCAP compliance

Default: `false`

Notes:

Example: `enableScap = false`

forceWalletReload

Syntax: `forceWalletReload = true|false`
Description: Defines whether or not the cache may be used for balance inquiries.
Type: Boolean
Optionality: Optional (default used if not set).
Allowed: true, false
Default: false
Notes:
Example: `forceWalletReload = true`

includeDcdCdrFields

Syntax: `includeDcdCdrFields = true|false`
Description: Whether or not to record the Result-Code in call EDRs.
Type: Boolean
Optionality: Optional (default used if not set).
Allowed: true, false
Default: false
Notes: See *DCD EDR Tags* (on page 93) for the list of DCD tags.
Example: `includeDcdCdrFields = false`

insufficientFundsDropCallResultCodes

Syntax:
```
insufficientFundsDropCallResultCodes = [
Integer
Integer
]
```
Description: Indicates a call drop immediately without granting any further time reservation including the withheld ones without sending back a CCR-T.
Type: Integer
Optionality: Optional
Allowed: Any predefined value
Default: None
Notes: insufficientFundsDropCallResultCodes parameter values are only enabled when voidUnusedReservation is set to false. If voidUnusedReservation set to true, insufficientFundsDropCallResultCodes parameter values are ignored and the default result codes, 4010 and 4012 are used.
Example:
```
insufficientFundsDropCallResultCodes = [
4013
4014
]
```

name

Syntax: `name = "type"`
Description: The name of the Domain Type as defined in Prepaid Charging.
Type: String
Optionality: Mandatory

Allowed: Defined in Prepaid Charging from available DIAMETER domain types on the **Domain** tab of the Service Management screen. Refer to *CCS User's Guide*.

Example: `name = "DIAMETER"`

overwriteZeroCallAnswerTime

Syntax: `overwriteZeroCallAnswerTime = true|false`

Description: Enables overwriting of Charging-Start-Timestamp using the EventTimestamp.

Type: Boolean

Optionality: Optional (default used if not set).

Allowed:
- true – Overwrite the charging start time.
- false – If a call is not answered (that is, CallAnswerTime is zero), set the charging start timestamp AVP to 'Jan 1, 1970 00:00:00.0000 UTC'.

Default: false

Notes:

Example: `overwriteZeroCallAnswerTime = false`

releaseOnLowCredit

Syntax: `releaseOnLowCredit = true|false`

Description: Whether to terminate a session after the expiry of the initial reservation when the reservation length is less than or equal to the low credit buffer. When `releaseOnLowCredit` is set to:
- true – DCD terminates sessions after the expiry of the initial reservation.
- false – DCD does not terminate the session.

Type: Boolean

Optionality: Required

Allowed: true, false

Default: false

Notes: When a call session using DCD with the UATB feature node is approaching a credit threshold, the UATB node needs enough usage units to provide an insufficient funds message. You should set `releaseOnLowCredit` to false if you have configured Diameter servers to assume that unused units are still available for the client. Otherwise, set `releaseOnLowCredit` to true to ensure that the client has enough unused units reserved for the unsufficient funds message.

The `releaseOnLowCredit` parameter should be placed immediately after the `voidUnusedReservation` parameter in the **eserv.config** file.

Example: `releaseOnLowCredit = false`

routing

Syntax: `routing = "`*`name`*`"`

Description: The algorithm to use when picking domains within the domain type.

Type: String

Optionality: Optional

Allowed:
- "Round Robin" (a weighted round robin algorithm)
- "Failover"

Default: "Round Robin"

Example: `routing = "Round Robin"`

`schemeName`

Syntax:	`schemeName = "name"`
Description:	The name of the peer scheme to use with this domain type.
Type:	String
Optionality:	Mandatory
Example:	`schemeName = "SchemeA"`

`voidUnusedReservation`

Syntax:	`voidUnusedReservation = true|false`
Description:	Whether or not to void unused reservations.
Type:	Boolean
Optionality:	Optional
Allowed:	true, false
Default:	false
Notes:	Diameter states that the server and client should consider any previously reserved, but unused, time in a subsequent reservation as no longer reserved. This is counter to most telephony models, so is disabled by default. To enable (for this domain type), set this parameter to true.
Example:	`voidUnusedReservation = false`

dynamicWalletReload

The `dynamicWalletReload` section defines the profile to use to determine whether to dynamically force wallet reloads. A wallet reload is forced when the `profileFormat` value is one of the following:

- "INTEGER" and the data in the profile is 1 (stored as 4-bytes)
- "BOOLEAN" and the data in the profile is 1 (stored as a single byte)
- "STRING" and the data in the profile is one of:
  - "true"
  - "yes"
  - "y"
  - "1"

The parameters are defined in the *ContextCopy Parameters* (on page 69) section.

Notes:

- Only the profile formats listed here are supported; all other values are ignored.
- The `forceWalletReload` parameter must be set to false (or not specified).

Example dynamicWalletReload

Here are the example parameters.

```
dynamicWalletReload = {
    profileBlock = 17
    profileTag = 6357900
    profileFormat = "INTEGER"
}
```

Domains Parameters

Here is an example of the `Domains` section.

```
Domains = [
```

```
        {
            name = "myDomainA"
            routing = "Round Robin"
            realmFailureWaitSeconds = 20
            weighting = 1
        }
        {
            name = "myDomainB"
            routing = "Failover"
            realmFailureWaitSeconds = 20
            weighting = 1
        }
    ]
```

name

Syntax: `name = "name"`

Description: The name of the domain as defined in Prepaid Charging.

Type: String

Optionality: Mandatory

Allowed: Defined in Prepaid Charging from available DIAMETER domains on the **Domain** tab of the Service Management screen. Refer to *CCS User's Guide*.

Example: `name = "myDomainA"`

realmFailureWaitSeconds

Syntax: `ArraySize = as`

Description: How long we wait for a Realm that has had a failure before retrying it.

Type: Integer

Optionality: optional

Allowed: in seconds

Default: 30

Example: `realmFailureWaitSeconds = 20`

routing

Syntax: `routing = "type"`

Description: The routing type to use when picking Realms within this Domain.

Type: String

Optionality: Optional

Allowed:

- "Round Robin" (a weighted round robin algorithm)
- "Failover"

Default: "Round Robin"

Example: `routing = "Round Robin"`

weighting

Syntax: `weighting = weight`

Description: The domain's weighting, if our Domain Type is using Round Robin routing.

Type: Integer

Optionality: Optional

Allowed:

Default: 1

Notes: The weighting determines how many times the domain is used. The weightings of all the active domains are added together and the domain receives its percentage of usage.

For example:

- myDomainA has a weighting of 1
- myDomainB a weighting of 1
- myDomainC a weighting of 2

myDomainA has 25% of the total usage. If myDomainC becomes unavailable, then myDomainA will have 50%.

Example: `weighting = 1`

AVPs

The `AVPs` section defines (as a tree-like structure) the configurable AVP part of the message for every action that uses Diameter requests and responses.

An example is provided in the **eserv.config.default** file.

The actions for which configuration is required are:

- BadPINRequest, BadPINResponse
- ConfirmNamedEventReservationRequest, ConfirmNamedEventReservationResponse
- ConfirmTimeReservationRequest, ConfirmTimeReservationResponse
- CreateEDRRequest, CreateEDRResponse
- DirectNamedEventRequest, DirectNamedEventResponse
- DirectTimeChargeRequest, DirectTimeChargeReponse
- ExtendTimeReservationRequest, ExtendTimeReservationReponse
- GetNamedEventRatesRequest, GetNamedEventRatesReponse
- InitialTimeReservationRequest, InitialTimeReservationReponse
- NamedEventReservationRequest, NamedEventReservationReponse
- RevokeNamedEventReservationRequest, RevokeNamedEventReservationReponse
- RevokeTimeReservationRequest, RevokeTimeReservationReponse
- WalletRechargeRequest, WalletRechargeResponse
- WalletInfoRequest, WalletInfoReponse

AVP Parameters

Here is an example of the `AVPs` section.

```
AVPs = [

    {
        name = "ConfirmNamedEventReservationRequest"
        mandatoryContents = [
            "Subscription-Id"
            "Service-Identifier"
            "NE-Used-Service-Unit"
        ]
        optionalContents = []
    }
    {
        avpCode = 443

        name = "Subscription-Id"
        type = "Grouped"
```

```
    mandatoryContents = [
        "Subscription-Id-Type"
        "Subscription-Id-Data"
    ]
    optionalContents = []
}
{
    avpCode = 450
    name = "Subscription-Id-Type"
    type = "Enumerated"

    literal = "0"           # END_USER_E164
}
{
    avpCode = 444
    name = "Subscription-Id-Data"
    type = "UTF8String"

    ccsConcept = "acsProfile"
    profileBlock =18
    profileTag = 327686
    profileFormat = "LNSTRING"
    conditionProfileBlock = 17
    conditionProfileTag = 2009
    conditionValue = 14
}
{
    avpCode = 439
    name = "Service-Identifier"
    type = "Unsigned32"
    ccsConcept = "eventType"
    cdrTag = "DIAMETER_SERVICE_ID"
    cdrOperation = "replace"
}
{
    avpCode = 446
    name = "NE-Used-Service-Unit"
    type = "Grouped"
    mandatoryContents = [
        "CC-Service-Specific-Units-Used"
    ]
}
{
    avpCode = 417
    name = "CC-Service-Specific-Units-Used"
    type = "Unsigned64"
    ccsConcept = "numUnitsUsed"

    vendorId = 0
    flags = 0
    repeating = false
    maxOccurrences = 10
    conversion = [
        {
            esg = 1
            vendor = 2
            serviceProvider = 1

        }
        {
            esg = 3
            vendor = 2
            serviceProvider = 2
```

```
        }
    ]

    conversionScale = 1
    conversionRounding = "floor"
    interpretBase = 10

    octetLength = 0
}
{
    avpCode = 900
    vendorId = 16247
    name = "Custom-Scp-Action"
    type = "Enumerated"
    ccsConcept = "scpAction"
    repeating = true
}

{
    name = "DirectNamedEventRequest-Money Refund"
    mandatoryContents = [
        "Subscription-Id"
        "SOS-TopUp-RequestedAction"
        "Charging-Start-Timestamp"
    ]
}

{
    name = "SOS-TopUp-RequestedAction"
    type = "Enumerated"
    value = "5"
    avpCode = 436
}

{
    name = "WalletRechargeRequest"
    mandatoryContents = [
        "Subscription-Id"
        "TopUp-RequestedAction"
        "Charging-Start-Timestamp"
        "Topup-Amount"
        "Topup-Voucher-Number"
        "Topup-Voucher-Type"
        "Topup-Voucher-Id"
        "Topup-Voucher-Serial-Number"
        "Source-System-Id"
    ]
    optionalContents = [
        "Voucher-Recharge_Failed-Flag"
        "Voucher-Recharge_Failed_Date_Time"
        "Topup-Voucher-Balance-Validity-Start"
        "Topup-Voucher-Balance-Validity-Relative"
    ]
}

{
    name = "Voucher-Recharge_Failed-Flag"
    type = "Integer32"
    vendorId = 581
    avpCode = 50998
    ccsConcept = "voucherRechargeFailureFlag"
    includeIf = true
}
```

```
{
    name = "Voucher-Recharge_Failed-Date-Time"
    type = "Integer32"
    vendorId = 581
    avpCode = 50999
    ccsConcept = "voucherRechargeFailureDateTime"
    excludeWhenIn = "0"
}

{
    name = "TopUp-RequestedAction"
    type = "Enumerated"
    value = "4"
    avpCode = 436
}

{
     name = "Charging-Start-Timestamp"
     type = "Integer32"
     profileBlock = 18
     profileTag = 327999
     avpCode = 12000
}

{
     name = "Topup-Voucher-Number"
     avpCode = 12001
     ccsConcept = "voucherInfoVoucher"
     type = "Integer32"
}

{
     name = "Topup-Amount"
     avpCode = 12002
     type = "Grouped"
     mandatoryContents = [
             "Value-Digits"
        ]
     optionalContents = [
         "Exponent"
     ]
}

{
    name = "Value-Digits"
    avpCode = 12003
    ccsConcept = "voucherInfoValue"
    type = "Integer32"
}

{
    name = "Topup-Voucher-Type"
    avpCode = 12005
    ccsConcept = "voucherTypeName"
    type = "UTF8String"
}

{
    name = "Topup-Voucher-Id"
    vendorId = 581
    avpCode = 50026
    ccsConcept = "voucherInfoVoucherId"
```

```
    type = "Integer32"
}

{

    name = "Topup-Voucher-Serial-Number"
    avpCode = 12008
    ccsConcept = "voucherInfoVoucherSerialNumber"
    type = "Unsigned64"
}

{
    name = "Topup-Voucher-Balance-Validity-Start"
    avpCode = 12009
    ccsConcept = "voucherInfoBalanceValidityStart"
    type = "Time"
}

{
    name = "Topup-Voucher-Balance-Validity-Relative"
    avpCode = 12010
    type = "Grouped"
    optionalContents = [
        "Topup-Voucher-Balance-Validity-Offset"
        "Topup-Voucher-Balance-Validity-Type
    ]
}

{

    name = "Topup-Voucher-Balance-Validity-Offset"
    avpCode = 12011
    ccsConcept = "voucherInfoBalanceValidityOffset"
    type = "Integer32"
}

{
    name = "Topup-Voucher-Balance-Validity-Type"
    avpCode = 12012
    ccsConcept = "voucherInfoBalanceValidityType"
    type = "Enumerated"
}

{
    name = "WalletRechargeResponse"
    mandatoryContents = [
        "Topup-Balance-Information"
        "Topup-Receipt-Number"
    ]
}

{
    name = "Topup-Balance-Information"
    vendorId = 581 # Intec
    avpCode = 50030
    type = "Grouped"
    mandatoryContents = [
        "Topup-Balance-Type-ID"
    ]
    optionalContents = [
        "Topup-Balance-Expire-Date"
        "Topup-Unit-Value"
    ]
}
```

```
{
    name = "Topup-Balance-Type-ID"
    avpCode = 50020
    vendorId = 581 # Intec
    ccsConcept = "walletInfoBalanceType"
    type = "Integer32"
}

{
    name = "Topup-Balance-Expire-Date"
    vendorId = 581 # Intec
    avpCode = 50032
    ccsConcept = "walletInfoBalanceExpiry"
    type = "Integer32"
}

{
    avpCode = 445
    name = "Unit-Value-Topup"
    type = "GroupedUnitValue"
    ccsConcept = "voucherInfoValue"
    conversionScale = -100
    signInversion = true
    mandatoryContents = [
        "Value-Digits-Topup"
    ]
    optionalContents = [
        "Exponent-Outgoing"
    ]
}

{
    avpCode = 447
    name = "Value-Digits-Topup"
    type = "Integer64"
}

{
    avpCode = 429
    name = "Exponent-Outgoing"
    type = "Integer32"
    literal = "1"
}

{
    name = "Topup-Value-Digits"
    avpCode = 50020
    vendorId = 581 # Intec
    ccsConcept = "walletInfoBalanceSystemValue"
    type = "Integer32"
}

{
     name = "Topup-Exponent"
     avpCode = 50199 # the ICD doesn't define this but it's not an issue
    for testing
     vendorId = 581 # Intec
     type = "Integer32"
}

{
    name = "Topup-Receipt-Number"
```

```
            avpCode = 50024
            type = "UTF8String"
        }

        {
            name = "CreateEDRRequest"
        }

        {
            name = "CreateEDRResponse"
        }


        {
            name = "BadPINRequest"
        }

        {
            name = "BadPINResponse"
        }

        {
            name = "Termination-Cause"
            avpCode = 295
            type = "Enumerated"
            ccsConcept = "terminationCause"
            conversion = [
                {
                    esg = 8 # releasedNoFunds
                    vendor = 6 # DIAMETER_AUTH_EXPIRED
                }
                {
                    esg = 9 # disconnectedLegBNoFunds
                    vendor = 6 # DIAMETER_AUTH_EXPIRED
                }

                {
                    esg = 14 # callingPartyDisconnected
                    vendor = 1 # DIAMETER_LOGOUT
                }
                {
                    esg = 15 # calledPartyDisconnected
                    vendor = 1 # DIAMETER_LOGOUT
                }
            ]
        }
]
```

avpCode

Syntax: `avpCode = code`

Description: The numeric tag code that is to be set whenever an AVP of this type is created (for example, added to a request message). It can also be used to ascertain the type of AVP unpacked from a response message.

Type: Integer

Optionality: Mandatory

Example: `avpCode = 888005`

ccsConcept

Syntax: `ccsConcept = "concept"`

Description: The “CCS concept” to which the AVP directly relates.

Type: String

Optionality: Optional. Used by some, though not all AVPs.

Allowed: The value is a string value, associated in the code with an enumeration. See *ccsConcepts* (on page 4).

Notes: The AVPs are ultimately filled out from available "ccsConcepts". These represent variables available to the DCD actions library at the time of sending the message.

Example: `ccsConcept = "acsProfile"`

cdrTag

Syntax: `cdrTag = "tag_name"`

Description: The EDR tag name to amend the EDR record with, depending on the cdrOperation parameter value.

Type: String

Optionality: Optional

Allowed: Alphanumeric characters only, plus underscore.

Default: None

Notes:

- cdrTag is optional, but if cdrOperation is specified then cdrTag must also be specified.
- For an AVP you can just specify cdrTag, then cdrOperation will default to "replace".
- The tag name should refer to a DCD tag, not one of the pre-defined ACS tags. Otherwise the operation will have no effect.

Example: `cdrTag = "CMX_EN"`

cdrOperation

Syntax: `cdrOperation = "operation"`

Description: The operation to perform on the cdrTag value in the EDR record.

Type: String

Optionality: Optional – only referred to if cdrTag is non-blank.

Allowed:

- replace
- leave
- append

Not case sensitive, for example Replace = REPLACE = replace

Default: replace

Notes:

- If cdrOperation is specified, cdrTag must also be specified.
- replace – If this tag is present, all instances are removed from the EDR and then append this instance to the EDR.
- leave – If this tag already exists, do nothing. Otherwise, append this instance to the EDR.
- append – Regardless of existence or not of this tag, append this instance to the EDR.

Example: `cdrOperation = "replace"`

conditionProfileBlock

Syntax: `conditionProfileBlock = block_number`

Description: The profile block to use for conditional AVP filling.

Type:	Integer
Optionality:	Optional (when omitted no condition checking is performed).
Allowed:	Any valid profile block number.
Default:	None
Notes:	If both the block and tag are specified, and there is no data in the location, then condition checking will fail and the AVP will *not* be populated.
Example:	`conditionProfileBlock = 17`

conditionProfileTag - 1.0.4 - 94934

Syntax:	`conditionProfileTag = tag_number`
Description:	The profile block field to use for conditional AVP filling.
Type:	Integer
Optionality:	Optional (missing then no condition checking is performed).
Allowed:	Any valid profile field number.
Default:	None
Notes:	If both the block and tag are specified, and there is no data in the location, then condition checking will fail and the AVP will *not* be populated.
Example:	`conditionProfileTag = 2009`

conditionValue

Syntax:	`conditionValue = condition`
Description:	The value to use to determine if AVP filling is to be performed.
Type:	Integer
Optionality:	Optional (default used if omitted).
Allowed:	Any positive integer.
Default:	1
Notes:	This example populates the AVP if and only if profile block/tag contains value 14.
Example:	`conditionValue = 14`

conversion

Syntax:	`conversion = [mapping]`
Description:	An array of integer values, defining a mapping from the Oracle enumeration to the billing vendor's equivalent enumeration value. The `serviceProvider` array parameter is optional, and it allows you to limit a conversion to a single service provider.
Type:	Integer
Optionality:	Mandatory
Notes:	• It is used if the AVP type is any of the integer types (including enumeration). This mapping is performed regardless of the ccsConcept, and is the responsibility of the AVP traverser, not the ccsConcept helper functions. • For AVPs associated with Balance Type concepts, do not include in the `conversion` array any balance types that are specified in the **Balance Type Mapping** tab of the SMS Service Management screen. Otherwise, the conversions from both sources could be applied.
Example:	`conversion = [` `  {`

```
            esg = 880
            vendor = 880880
            serviceProvider = 1
        }
        {
            esg = 890
            vendor = 890890
            serviceProvider = 2
        }
    ]
```

esg

Syntax: `esg = int`

Description: The integer value in the conversion array that indicates which equivalent value should be used by the billing vendor.

Type: Integer

Optionality: Mandatory

Allowed:

Default:

Notes: This value differs based on the AVP that uses it.

For example, see *ACS Action handler* (on page 5) for `esg` values defined for the `Termination-Cause` AVP in the configuration.

Example:

```
esg = 3
```

vendor

Syntax: `vendor = int`

Description: The billing vendor's integer value in the conversion array that maps to the equivalent `esg` value.

Type: Integer

Optionality: Mandatory

Allowed:

Default:

Notes: This value differs based on the AVP that uses it.

For example, the vendor values defined for the `Termination-Cause` AVP in the configuration are as follows:

Integer	Reason	Comment
1	DIAMETER_LOGOUT	The user initiated a disconnect
2	DIAMETER_SERVICE_NOT_PROVIDED	This value is used when the user disconnected prior to the receipt of the authorization answer message.
3	DIAMETER_BAD_ANSWER	This value indicates that the authorization answer received by the access device was not processed successfully.
4	DIAMETER_ADMINISTRATIVE	The user was not granted access, or was disconnected, due to administrative reasons, such as the receipt of a Abort-Session-Request message.
5	DIAMETER_LINK_BROKEN	The communication to the user was abruptly disconnected.
6	DIAMETER_AUTH_EXPIRED	The user's access was terminated since its authorized session time has expired.
7	DIAMETER_USER_MOVED	The user is receiving services from another access device.
8	DIAMETER_SESSION_TIMEOUT	The user's session has timed out, and service has been terminated.

Example: `vendor = 6`

`serviceProvider`

Syntax: `serviceProvider = int`

Description: If mentioned, the conversion is limited to the specific service provider.

Type: Integer

Optionality: Optional

Allowed:

Default:

Notes:

Example: `serviceProvider = 2`

`conversionScale`

Syntax: `conversionScale = scale`

Description: Defines a conversion factor of `esg` values to calculate server values.

Type: Integer

Optionality: Optional

Allowed:
- 0 – Applies the scale factor specified in the **Balance Type Mapping** tab of the SMS Service Management screen. If the **Balance Type Mapping** tab does not contain an applicable mapping, DCD applies a scale factor of 1. For more information, see *CCS User's Guide*.
- Any non-zero integer – Applies the scale factor to all instances of the AVP in request and response messages. For example, if you set `conversionScale` to 100, DCD multiplies the values by 100 for all balance type AVPs.

Default: 1

Notes:
- For request AVPs – Positive means multiply, negative means divide.
- For response AVPs – Positive means divide, negative means multiply.

All conversion rules are applied before scaling is applied.

Example: `conversionScale = -10`

This example multiplies incoming Diameter values by 10.

`conversionRounding`

Syntax: `conversionRounding = "`*`rounding_type`*`"`

Description: The conversion method used between internal and server numeric values.

Type: String

Optionality: Optional (default used if not set).

Allowed:
- floor – Drop any fractions.
- ceiling – Round up fractional parts.
- round – Round to the nearest whole number. That is, x.5 or higher is rounded up and others are rounded down.

Default: floor

Notes:
- If an AVP has the `conversionScale` parameter set, `conversionRounding` can also be set.
- For GroupedUnitValue AVP types, use the `conversionRounding` parameter to specify the type of rounding applied after applying an exponent value.

Example: `conversionRounding = "round"`

`excludeIf`

Syntax: `excludeIf = true|false`

Description: DCD outgoing messages will exclude AVP values that match this rule.

Type: Boolean

Optionality: Optional.

Allowed: true, false

Default: N/A

Notes: The AVP `type` should be set to "Integer32".

If `includeIf` is also defined then DCD logs a warning that the configuration is inconsistent. Depending on which parameter appears first in the configuration file DCD will load either `excludeIf` or `includeIf`, but not both.

Example: `excludeIf = false`

excludeIfMatches

Syntax: `excludeIfMatches = "search_string"`

Description: DCD outgoing messages will exclude AVP values that match this rule. The AVP type should be set to "UTF8String".

Type: String

Optionality: Optional

Allowed:

"*search_string*"	Check for the specified search string anywhere in the string
"^*search_string*$"	Check the specified search string matches the whole string
"^*search_string*"	Check for the specified search string at the beginning of the string
"*search_string*$"	Check for the specified search string at the end of the string

Default: N/A

Notes: If `includeIfMatches` is also defined then DCD logs a warning that the configuration is inconsistent. Depending on which parameter appears first in the configuration file DCD will load either `excludeIfMatches` or `includeIfMatches`, but not both.

Example: `excludeIfMatches = "f006$"`

excludeWhenIn

Syntax: `excludeWhenIn = "range|list"`

Description: DCD outgoing messages will exclude AVP values that match this rule.

Type: Integer

Optionality: Optional .

Allowed:

delimited range (numbers from x to y)	"*x..y*"
greater than or equal to x	">=*x*"
less than or equal to x	"<=*x*"
any of the listed numbers x, y or z (list may be any length)	"*x,y,z*"
a single number x	"*x*"

Default: N/A

Notes: The AVP `type` should be set to "Integer32".

If `includeWhenIn` is also defined then DCD logs a warning that the configuration is inconsistent. Depending on which parameter appears first in the configuration file DCD will load either `excludeWhenIn` or `includeWhenIn`, but not both.

Example: `excludeWhenIn = "1..99"`

flags

Syntax: `flags = number`

Description: What flags to override in the AVP header (as octet value). For example, for M, V bits: flags = 192

Type: Integer

Optionality: Optional (default used if not set).

Allowed:

Default: 0

Notes: If the `vendorID` parameter value is greater than zero, then V-bit will be set regardless.

Example:
```
flags = 192
```

includeIf

Syntax:
```
includeIf = true|false
```

Description: DCD outgoing messages will include AVP values that match this rule.

Type: Boolean

Optionality: Optional.

Allowed: true, false

Default:

Notes: The AVP `type` should be set to "Integer32".

If `excludeIf` is also defined then DCD logs a warning that the configuration is inconsistent. Depending on which parameter appears first in the configuration file DCD will load either `excludeIf` or `includeIf`, but not both.

Example:
```
includeIf = true
```

includeIfMatches

Syntax:
```
includeIfMatches = "search_string"
```

Description: DCD outgoing messages will include AVP values that match this rule.

Type: Integer

Optionality: Optional .

Allowed:

search_string	Check for the specified search string anywhere in the string
^*search_string*$	Check the specified search string matches the whole string
^*search_string*	Check for the specified search string at the beginning of the string
search_string$	Check for the specified search string at the end of the string

Default: N/A

Notes: The AVP `type` should be set to "UTF8String".

If `excludeIfMatches` is also defined then DCD logs a warning that the configuration is inconsistent. Depending on which parameter appears first in the configuration file DCD will load either `excludeIfMatches` or `includeIfMatches`, but not both.

Example:
```
includeIfMatches = "^f003"
```

includeWhenIn

Syntax:
```
includeWhenIn = "range|list"
```

Description: DCD outgoing messages will include AVP values that match this rule.

Type: Integer

Optionality:	Optional.	
Allowed:	delimited range (numbers form x to y)	"*x..y*"
	greater than or equal to x	">=*x*"
	less than or equal to x	<=*x*"
	any of the listed numbers x, y or z (list may be any length)	"*x*,*y*,*z*"
	a single number x	"*x*"
Default:	N/A	
Notes:	If `includeWhenIn` is also defined, then DCD logs a warning that the configuration is inconsistent. Depending on which parameter appears first in the configuration file DCD will load either `excludeWhenIn` or `includeWhenIn`, but not both.	
Example:	`includeWhenIn = "12,14,16,-18,20,22"`	

interpretBase

Syntax: `interpretBase = number`

Description: Defines the base to use when interpreting numbers that are stored as strings.

Type: Integer

Optionality: Optional (default used if not set).

Allowed:

- 0 – See notes
- 8 – Octal
- 10 – Decimal
- 16 – Hexadecimal

Default: 10 (decimal)

Notes: If the configured value is 0, strings are interpreted in the order of decimal constant, octal constant, or hexadecimal constant. Any of these may be preceded by a + or a – sign.

- Decimal constant – Begins with a non-zero digit and consists of a sequence of decimal digits.
- Octal constant – Begins with a 0 (zero) followed by a sequence of the digits 0 to 7.
- Hexadecimal constant – Begins with a 0x or 0X followed by a sequence of the decimal digits and letters a (or A) to f (or F).

Example: `interpretBase = 10`

literal

Syntax: `literal = "value"`

Description: A literal value.

Type: String

Optionality: Optional

Default: empty

Notes:

- In some cases where we use the AVP definition to create a request message, we may wish to simply specify a literal value rather than obtain the information from a ccsConcept field. In these cases we specify the value here, in string form, and it will be converted to the required type when requested from the configuration AVP object.

- This parameter was previously "value".

Example:
```
literal = "1"
```

mandatoryContents

Syntax: `mandatoryContents = "avps"`

Description: A simple string array, applicable to AVPs of type “Grouped” and those with no type, specifying the AVPs (by name) that must be included in this AVP

Type: String array

Optionality: Optional

Default: empty

Example:
```
mandatoryContents = [
    "Subscription-Id-Type"
    "Subscription-Id-Data"
]
```

maxOccurrences

Syntax: `maxOccurrences = value`

Description: The number of repeating AVPs (whether a group or an individual field).

It does not actually limit how many AVPs can be processed; it limits how the AVP is physically 'unloaded' into a contiguous range of profile variables.

Type: Integer

Optionality: Optional (default used if not set).

Allowed:

Default: 0 – The default value means that all AVP values are unloaded into the same configured profile block/tag (admittedly not very useful, but this ensures backwards compatibility).

Notes: To unload an array of AVPs into a *series* of profile tag variables `maxOccurrences` is set to the number of different values to be captured.

The first will be unloaded into the profile tag configured for the AVP.

The second will be unloaded into that tag+1, and so on.

Note: When unloading a repeating group the specially-defined variable 'index' can be obtained by coding an AVP with variable="index". This takes the value of the current loop index.

Example:
```
maxOccurrences = 10
```

name

Syntax: `name = "name"`

Description: The name of the AVP, which can be used by the code for direct retrieval, as well as logging and debug messages.

Type: String

Optionality: Mandatory

Example:
```
name = "CC-Money"
```

octetLength

Syntax: `octetLength = value`

Description: How many bytes to use for integer quantities, if the `type` is "OctetString".

Type:	Integer
Optionality:	Optional
Allowed:	Should be a number 0 through 4.
Notes:	0 is a special case meaning encode the value as an ASCII string, and determine the number of bytes from the string size. For responses the values 1-4 simply mean treat as integer encoded, and use the number of bytes given.
Example:	`octetLength = 0`

`optionalContents`

Syntax:	`optionalContents = "avps"`
Description:	A simple string array, applicable to AVPs of type “Grouped” and those with no type, specifying the AVPs (by name) that may be included in this AVP
Type:	String
Optionality:	Optional
Default:	empty

Example:	`optionalContents = [`
	`  "Service-Identifier"`
	`  "Requested-Service-Unit"`
	`  "Subscription-Id"`
	`]`

`profileBlock`

Syntax:	`profileBlock = num`
Description:	The profile block
Type:	Integer
Optionality:	Used only if the ccsConcept field is a profile variable.
Allowed:	The value given for this is a number, and must be valid for a profile block (that is, in the range 0 through 18).
Notes:	This parameter is used to identify the profile block it will be stored into/retrieved from. See also the related parameters, `profileTag` and `profileFormat`.
Example:	`profileBlock = 9`

`profileFormat`

Syntax:	`profileFormat = "format"`
Description:	The format of the profile.
Type:	String
Optionality:	Used only if the `ccsConcept` field is a profile variable.
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: • STRING • NSTRING • LNSTRING • INTEGER • UNSIGNED64 • RAW • TIME • BOOLEAN • ARRAY

Default: INTEGER

Notes: This parameter is used to identify its storage format. See also the related parameters, `profileBlock` and `profileTag`.

Example: `profileFormat = "LNSTRING"`

`profileTag`

Syntax: `profileTag = num`

Description: The profile tag.

Type: Integer

Optionality: Used only if the `ccsConcept` field is a profile variable.

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

Example: `profileTag = 999`

`repeating`

Syntax: `repeating = true|false`

Description: Allows the configuration-driven code to recognize that there may be a number of repeating instances of this AVP in its containing group, not just one.

Type: Boolean

Optionality: Mandatory

Allowed: true, false

Notes: In the building of a request, repeating AVPs are added until the getConcept call indicates no more concept data is available. In the unpacking of a response, repeating AVPs are extracted (and setConcept calls made) until the Diameter stack indicates there are no more to retrieve.

Example: `repeating = true`

`signInversion`

Syntax: `signInversion = true | false`

Description: When this parameter is true it converts the value from positive to negative and vice versa for AVP types of Integer32 and Integer64. If this parameter is true for an AVP then:

- An outbound positive DCD concept value will be converted to a negative value in the AVP.
- An outbound negative DCD concept value will be converted to positive value in the AVP.
- An inbound positive AVP value will be converted to a negative DCD concept value.
- An inbound negative AVP value will be converted to a positive DCD concept value.

Type: Boolean

Optionality: Optional (default used if not set)

Allowed: true, false

Default: false

Notes:

Example: `signInversion = true`

`type`

Syntax: `type = "type"`

Description: The type of AVP.

Type: String

Optionality: Optional. When defining the AVPs that make up a request message, you can list them as mandatory or optional contents in an AVP that has no other information. This means the contents should be directly placed into the request message without (for example) an intervening group.

Allowed: If specified, this string value must be the name of any previously defined AVP in the configuration, or one of the following base types that are described in the Diameter RFC 3588 specification:

- OctetString
- Integer32
- Integer64
- Unsigned32
- Unsigned64
- Grouped
- GroupedUnitValue
- Address
- Time
- UTF8String
- DiameterIdentity
- DiameterURI
- Enumerated

Note: Float32 and Float64, although defined in RFC 3588, are not supported because CCS does not use floating point values.

Notes: Specify the name of a previously defined AVP when you want to relate two CCS concept fields to the same base type without having to repeat the full definition of that base type. All the attributes of the base type are inherited except the base type name, repeating attribute and ccsConcept value.

Example: `type = "Grouped"`

`vendorId`

Syntax: `vendorId = ID`

Description: A number that identifies the vendor ID of the corresponding Diameter AVP.

Type: Integer

Optionality: Optional

Default: 0

Notes: It will be used whenever we have to insert this AVP into a request message.

Example: `vendorId = 0`

Example Configuration for an AVP Type of GroupedUnitValue

This section shows how DCD converts balance-related AVPs with a type of GroupedUnitValue. For more information, see *type* (on page 66).

Example Configuration for Request Messages:

The following example configuration specifies to perform the following for outgoing request messages with an AVP type of GroupedUnitValue:

- Divide the value by 100
- Apply sign inversion

```
{
    avpCode = 445
    name = "Unit-Value-Topup"
    type = "GroupedUnitValue"
    ccsConcept = "voucherInfoValue"
    conversionScale = -100
    signInversion = true
    mandatoryContents = [
        "Value-Digits-Topup"
    ]
    optionalContents = [
        "Exponent-Outgoing"
    ]
}
{
    avpCode = 447
    name = "Value-Digits-Topup"
    type = "Integer64"
}
{
    avpCode = 429
    name = "Exponent-Outgoing"
    type = "Integer32"
    literal = "1"
}
```

For example, if a voucher top-up in NCC has a value of -2000, DCD converts it to 20 after applying scaling and sign inversion. In this case, the GroupedUnitValue AVP in the outgoing request message to the third-party application would have Value-Digits set to 2 and Exponent set to 1.

Note: An exponent is always sent for GroupedUnitValue AVPs. If the `literal` parameter is not defined, it defaults to 0.

Example Configuration for Response Messages:

The following example configuration specifies to apply sign inversion to incoming response messages with an AVP type of GroupedUnitValue:

```
{
    avpCode = 252
    name = "ORA-Credit-Floor"
    vendorId = 3512
    type = "GroupedUnitValue"
    ccsConcept = "walletInfoBalanceMaxCredit"
    signInversion = true
    mandatoryContents = [
        "Value-Digits-Credit-Floor"
    ]
    optionalContents = [
        "Exponent-Incoming"
    ]
}
{
    avpCode = 447
    name = "Value-Digits-Credit-Floor"
    type = "Integer64"
}
```

```
{
    avpCode = 429
    name = "Exponent-Incoming"
    type = "Integer32"
}
```

For example, if an incoming response message from a third-party application contains a GroupedUnitValue AVP with Value-Digits set to 5 and Exponent set to 3, DCD converts the credit floor value to -5000 after applying sign inversion.

Note: If the exponent is not supplied in the incoming answer message, DCD applies an exponent of 0 to meet RFC 4006 guidelines.

Example cdrTag/Operation Configuration

The `cdrTag` and `cdrOperation` parameters can be used by any AVP/ccsConcepts pairing.

The following is just an example of how to configure these parameters.

```
{
    # This 'AVP' simply defines the list of AVPs for a direct named event
    reservation.

    # Give this AVP a position in the configuration 'tree'. This is just
    # a label to allow representation of the tree to the actions library.
    name = "DirectNamedEventRequest"

    # Define the list of AVPs (by name) for this node. Note that the AVPs are at the
    # sibling level for this node if there are no type or avpCode parameters for this
    node.
    # An error will occur if mandatory contents are not available, but non-present
    optional
    # contents are silently ignored.
    mandatoryContents = [
        "Service-Identifier"
        "ChargingMaxEventClassAndEventName"
    ]
    optionalContents = []
}
{

    avpCode = 13000
    name = "ChargingMaxEventClassAndEventName"
    type = "Grouped"
    mandatoryContents = [
        "CMX-eventClass"
        "CMX-eventName"
        "DIA-Service-Identifier"
    ]
    optionalContents = []
}
{

    avpCode = 13001
    name = "CMX-eventClass"
    ccsConcept = "eventClass"
    type = "UTF8String"
    cdrTag = "CMX_EC"
    cdrOperation = "append"
}
{

    avpCode = 13002
    name = "CMX-eventName"
```

```
    ccsConcept = "eventName"
    type = "UTF8String"
    cdrTag = "CMX_EN"
    cdrOperation = "leave"
}
{
    # This 'AVP' represents the RFC 4006 Service-Identifier.
    avpCode = 439
    name = "DIA-Service-Identifier"
    type = "Unsigned32"
    ccsConcept = "eventType"
    cdrTag = "DIA_SI"
    cdrOperation = "replace"
}
...
```

ContextCopy Parameters

The ContextCopy section defines the profiles in which DCD call context data may be copied to at call time. This is normally for the purposes of control plan branching.

Example ContextCopy Section

Here is an example of the `ContextCopy` section.

```
ContextCopy = [
{
    contextItem = "scpActionSupervise"
    profileBlock = 17
    profileTag = 6356992 # Hex 0x00610000
    profileFormat = "INTEGER"
}
{
    contextItem = "scpActionDoNotSupervise"
    profileBlock = 17
    profileTag = 6356993 # Hex 0x00610001
    profileFormat = "INTEGER"
}
{
    contextItem = "scpActionRelease"
    profileBlock = 17
    profileTag = 6356994 # Hex 0x00610002
    profileFormat = "INTEGER"
}
{
    contextItem = "scpActionSendMessage"
    profileBlock = 17
    profileTag = 6356995 # Hex 0x00610003
    profileFormat = "INTEGER"
}
{
    contextItem = "scpActionPlayAnnouncement"
    profileBlock = 17
    profileTag = 6356996 # Hex 0x0061004
    profileFormat = "INTEGER"
}
{
    contextItem = "scpActionSuperviseWithoutControlling"
    profileBlock = 17
    profileTag = 6356997 # Hex 0x0061005
    profileFormat = "INTEGER"
}
```

```
        {
            contextItem = "callState"
            profileBlock = 17
            profileTag = 6356998 # Hex 0x0061006
            profileFormat = "INTEGER"
        }
        {
            contextItem = "sendCount"
            profileBlock = 17
            profileTag = 6356999 # Hex 0x0061007
            profileFormat = "INTEGER"
        }
        {
            contextItem = "preCallAnnouncementId"
            profileBlock = 17
            profileTag = 6357000 # Hex 0x0061008
            profileFormat = "INTEGER"
        }
        {
            contextItem = "preCallLowBalance"
            profileBlock = 17
            profileTag = 6357001 # Hex 0x0061009
            profileFormat = "INTEGER"
        }
        ]
```

contextItem

Syntax: `contextItem = "name"`

Description: The name of the DCD context item from the allowed list.

Type: String

Optionality: Optional

Allowed:

- "scpActionSupervise"
- "scpActionDoNotSupervise"
- "scpActionRelease"
- "scpActionSendMessage"
- "scpActionPlayAnnouncement"
- "scpActionSuperviseWithoutControlling"
- "callState"
- "sendCount"
- "preCallAnnouncementId"
- "preCallLowBalance"

Default:

Notes: All the supported context items are listed in the example.

Example: `contextItem = "scpActionSupervise"`

profileBlock

Syntax: `profileBlock = val`

Description: The profile block to use.

Type: Integer

Optionality: Optional (default used if not set).

Allowed:

Default:

Notes: Examples all use 17 (temporary storage)

Example:	`profileBlock = 17`

`profileFormat`

Syntax:	`profileFormat = "format"`
Description:	The profile tag format
Type:	String
Optionality:	Optional (default used if not set).
Allowed:	"INTEGER" "STRING" "TIME"
Default:	
Notes:	
Example:	`profileFormat = "INTEGER"`

`profileTag`

Syntax:	`profileTag = decival`
Description:	The profile tag in which to store data.
Type:	Integer
Optionality:	Optional (default used if not set).
Allowed:	
Default:	
Notes:	Example tag values are all in the DCD range (0x0061nnnn), but will need to be configured in ACS and Prepaid Charging before they are available for use. In the example, 6356992 is value of Hex 0x00610000
Example:	`profileTag = 6356992`

TimeIn and TimeOut

Use TimeIn and TimeOut to copy a timestamp at the beginning of a call (TimeIn) and to record elapsed time during a call (TimeOut). If present, the timestamps are stored in the configured profile fields.

- TimeIn – The time the CCA was received by the SLC from the VWS.
- TimeOut – The time the CCR was sent from the SLC to the VWS.

These parameters are optional. If these parameters are omitted, no timestamps are recorded.

Example tag values are all in the DCD range (0x0061nnnn), but will need to be configured in ACS and Prepaid Charging before they are available for use.

The parameters are defined in the *ContextCopy Parameters* (on page 69) section.

Example TimeIn and TimeOut

Here are the example parameters.

```
TimeIn = {
    profileBlock = 17
    profileTag = 6357002 # Hex 0x006100a, continuing from example ContextCopy
    values
    profileFormat = "TIME"
}
TimeOut = {
```

```
            profileBlock = 17
            profileTag = 6357003 # Hex 0x006100b
            profileFormat = "TIME"
        }
```

Routes

Introduction

The `Routes` section is used to specify the routing configuration for the BeClient, that is, how to select a peer of the realm.

Routes Parameters

Here is an example routes section of the `DIAMETER` section of the **eserv.config** file.

```
    routes = [
        {
            realm = "FirstRealm"
            host = "host1.realm1.oracle.com"

            priority = 1
            round_robin = 0
            direct = true
        }
    ]
```

direct

Syntax:	`direct = true|false`
Description:	Whether this is a direct server connection, or if a proxy/agent is used.
Type:	Boolean
Optionality:	Optional
Allowed:	true, false
Default:	true
Notes:	May be specified for each host. See *HostSpecificData Parameters* (on page 73).
Example:	`direct = true`

host

Syntax:	`host = "`*`permitted_origin_host`*`"`
Description:	The host name of the next-hop as configured in `DCD.Peers.permittedOriginHosts` for the relevant peer.
Type:	String
Optionality:	Mandatory
Allowed:	This is the permittedOriginHosts value of the peer. May be specified for each host. See *HostSpecificData Parameters* (on page 73).
Example:	`host = "host1.realm1.oracle.com"`

priority

Syntax:	`priority = `*`priority`*
Description:	The priority of the route.
Type:	Integer
Optionality:	Optional

Notes:	Only those routes with the lowest priority are used. May be specified for each host. See *HostSpecificData Parameters* (on page 73).
Example:	`priority = 1`

`realm`

Syntax:	`realm = = name`
Description:	The Realm identity.
Type:	String
Optionality:	Mandatory
Allowed:	As configured in the **Domain** tab of the Service Management screen. Refer to *CCS User's Guide* for details. May be specified for each host. See *HostSpecificData Parameters* (on page 73).
Example:	`realm = "FirstRealm"`

`round_robin`

Syntax:	`round_robin = weight`
Description:	The weight for round_robin selection.
Type:	Integer
Optionality:	Optional
Default:	0
Notes:	Zero indicates a failover type selection. See *routing* (on page 45) for details. May be specified for each host. See *HostSpecificData Parameters* (on page 73).
Example:	`round_robin = 0`

HostSpecificData

Introduction

The following is an optional section, with members defined according to hostname. The purpose is to represent data specific to particular hosts. This permits a common configuration file to be deployed to multiple machines.

Settings here should override global settings, for the specified host only. See *DCD Parameters* (on page 33) and *Routes Parameters* (on page 72) for global settings of these parameters.

HostSpecificData Parameters

Here is an example of the `HostSpecificData` parameters.

```
HostSpecificData = [
    {
        name = "ocpc.oracle.com"


        DCD = {
            Origin-Host = "ocpc.oracle.com"
            Origin-Realm = "ocpc.oracle.com"
        }

        routes = [
```

```
                {
                    realm = "myDomainA"
                    host = "host1.realm1.oracle.com"

                    priority = 1
                    round_robin = 0
                    direct = true
                }
            ]

        }

    ]
```

direct

Syntax: `direct = true|false`
Description: Whether this is a direct server connection, or if a proxy/agent is used.
Type: Boolean
Optionality: Optional
Allowed: true, false
Default: true
Notes: See *Routes Parameters* (on page 72) for the global setting.
Example: `direct = true`

host

Syntax: `host = "name"`
Description: `This` is the host name of the next-hop.
Type: String
Optionality: Mandatory
Allowed: This is the permittedOriginHosts value of the peer.
Notes: See *Routes Parameters* (on page 72) for the global setting.
Example: `host = "host1.realm1.oracle.com"`

name

Syntax: `name = "hostname"`
Description: The identifier correlating to machine hostname (SLC node).
Type: String
Optionality: Mandatory if the optional `HostSpecificData` section is defined.
Allowed:
Default:
Notes:
Example: `name = "ocpc.oracle.com"`

Origin-Host

Syntax: `Origin-Host = "diameterId"`
Description: The Diameter AVP Origin-Host - fully qualified domain name.
Type: String
Optionality: Optional (default used if not set).
Allowed:

Default:
Notes: See *DCD Parameters* (on page 33) for the global setting.
Example: `Origin-Host = "ocpc.oracle.com"`

Origin-Realm

Syntax: `Origin-Realm = "diameterId"`
Description: The Diameter AVP Origin-Realm.
Type: String
Optionality: Optional (default used if not set).
Allowed:
Default:
Notes: See *DCD Parameters* (on page 33) for the global setting.
Example: `Origin-Realm = "ocpc.oracle.com"`

priority

Syntax: `priority = priority`
Description: This is the priority of the route.
Type: Integer
Optionality: Optional
Notes: Only those routes with the lowest priority are used.
See *Routes Parameters* (on page 72) for the global setting.
Example: `priority = 1`

realm

Syntax: `realm = "realmname"`
Description: The realm name, as configured in the Prepaid Charging screens.
Type: String
Optionality: Mandatory
Allowed:
Default:
Notes: See *Routes Parameters* (on page 72) for the global setting.
Example: `realm = "myDomainA"`

round_robin

Syntax: `round_robin = weight`
Description: This is the weight for round_robin selection.
Type: Integer
Optionality: Optional
Default: 0
Notes: Zero indicates a failover type selection. See *routing* (on page 45) for details.
See *Routes Parameters* (on page 72) for the global setting.
Example: `round_robin = 0`

NamedEventTypes

Introduction

The `NamedEventTypes` section is used to define a mapping from the Prepaid Charging descriptor of a named event (the `eventClass` and `eventName`) to a single integer (`eventType`) for availability to the Service-Identifier AVP.

NamedEventTypes Parameters

Here is an example of the NamedEventTypes section.

```
NamedEventTypes = [
    {
        eventClass = "abc"
        eventName = "def"
        eventType = 123
        isDebit = true
    }
    {
        eventClass = "ghi"
        eventName = "jkl"
        eventType = 456
    }
]
```

`eventClass`

Syntax:	`eventClass = "`*`class`*`"`
Description:	The event class.
Type:	String
Optionality:	Optional
Allowed:	Defined on the **Named Event** tab on the Rating Management screen. Refer to *CCS User's Guide* for details.
Example:	`eventClass = "abc"`

`eventName`

Syntax:	`eventName = "`*`name`*`"`
Description:	The event name.
Type:	String
Optionality:	Optional
Allowed:	Defined on the **Named Event** tab on the Rating Management screen. Refer to *CCS User's Guide* for details.
Example:	`eventClass = "def"`

`eventType`

Syntax:	`eventType =` *`type`*
Description:	The event type for availability to the Service-Identifier AVP.
Type:	integer
Optionality:	Optional
Allowed:	Mapping to agreed event type with server vendor.
Example:	`eventType = 123`

`isDebit`

Syntax:	`isDebit = true|false`
Description:	Whether this named event represents a debit or credit for the subscriber.
Type:	Boolean
Optionality:	Optional
Allowed:	true, false
Default:	true
Example:	`isDebit = true`

Chapter 5

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.

diameterBeClient

Purpose

The diameterBeClient takes SLEE messages from the diamActions library and converts them to Diameter messages. It also maintains all Diameter connections.

Startup

The diamActions library and diameterBeClient will load the configuration (from the **eserv.config** file and the SLC database) on startup and on receiving a SIGHUP, at which point it needs to reload the config.

Example DIAMETER Section

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

```
DIAMETER = {

    DCD = {
        serviceDomainInterfaceName = "dcdBeClient"
        loggedNotificationPeriod = 300
        loggedInvalidPeriod = 300

        databaseCacheValiditySeconds = 18000

        Origin-Host = "ocpc.oracle.com"

        Origin-Realm = "ocpc.oracle.com"

        serviceContextID = "ocpc@oracle.com"

        Auth-Application-Id = 4

        Vendor-Specific-Application-Id = [
            {
```

```
            Vendor-Id = 193
            Auth-Application-Id = 4
            Acct-Application-Id = 19302
            }
        ]

        enableDraft8 = false

        # scheme = "SchemeA"

        honour_disconnect = true

        Vendor-Id = 16247

        Product-Name = ""

        timeout_x = 30000000

        transmit_limit = 2

        trace_debug_flags = "all"

        statistics_interval = 300

        NotificationFilter = {
            CollectiveNotice = true
            PerPeerNotice = {
                initial_request = true
                update_request = true
                termination_request = true
                event_request = true
                unknown_request = true
                total_request = true
                request_timeout = true
                non_request_type_result_codes = true
            }

        }

    }

    DomainTypes = [
    {
        name = "DIAMETER"

        schemeName = "SchemeA"

        routing = "Round Robin"

        voidUnusedReservation = false

        defaultSessionFailover = 0
        defaultEventFailover = 0
        defaultFailureHandling = 0

        balanceEnquiryMethod = "balanceCheck"

        includeDcdCdrFields = false

        defaultFixedCostDuration = 86400
        enableScap = false
        overwriteZeroCallAnswerTime = false

        Domains = [
```

```
{
    name = "myDomainA"
    routing = "Round Robin"
    realmFailureWaitSeconds = 20
    weighting = 1
}
{
    name = "myDomainB"
    routing = "Failover"
    realmFailureWaitSeconds = 20
    weighting = 1
}
]

AVPs = [

    {
        name = "ConfirmNamedEventReservationRequest"
        mandatoryContents = [
            "Subscription-Id"
            "Service-Identifier"
            "NE-Used-Service-Unit"
        ]
        optionalContents = []
    }
    {
        avpCode = 443

        name = "Subscription-Id"
        type = "Grouped"
        mandatoryContents = [
            "Subscription-Id-Type"
            "Subscription-Id-Data"
        ]
        optionalContents = []
    }
    {
        avpCode = 450
        name = "Subscription-Id-Type"
        type = "Enumerated"

        literal = "0"           # END_USER_E164
    }
    {
        avpCode = 444
        name = "Subscription-Id-Data"
        type = "UTF8String"

        ccsConcept = "acsProfile"
        profileBlock =18
        profileTag = 327686
        profileFormat = "LNSTRING"
        conditionProfileBlock = 17
        conditionProfileTag = 2009
        conditionValue = 14
    }
    {
        avpCode = 439
        name = "Service-Identifier"
        type = "Unsigned32"
        ccsConcept = "eventType"
        cdrTag = "DIAMETER_SERVICE_ID"
        cdrOperation = "replace"
```

```
}
{
    avpCode = 446
    name = "NE-Used-Service-Unit"
    type = "Grouped"
    mandatoryContents = [
        "CC-Service-Specific-Units-Used"
    ]
}
{
    avpCode = 417
    name = "CC-Service-Specific-Units-Used"
    type = "Unsigned64"
    ccsConcept = "numUnitsUsed"

    vendorId = 0
    flags = 0
    repeating = false
    maxOccurrences = 10
    conversion = [
        {
            esg = 1
            vendor = 2
            serviceProvider = 1

        }
        {
            esg = 3
            vendor = 2
            serviceProvider = 2

        }
    ]

    conversionScale = 1
    conversionRounding = "floor"
    interpretBase = 10

    octetLength = 0
}
{
        name = ”3GPP-MS-TimeZone”
        vendorId = 10415 #3GPP
        avpCode = 23
        ccsConcept = “callerMsTimeZone”
        type = “OcterString”
        octetLength = 2
}

{
    avpCode = 900
    vendorId = 16247
    name = "Custom-Scp-Action"
    type = "Enumerated"
    ccsConcept = "scpAction"
    repeating = true
}

{
    name = "DirectNamedEventRequest-Money Refund"
    mandatoryContents = [
        "Subscription-Id"
        "SOS-TopUp-RequestedAction"
        "Charging-Start-Timestamp"
```

```
    ]
}

{
    name = "SOS-TopUp-RequestedAction"
    type = "Enumerated"
    value = "5"
    avpCode = 436
}

{
    name = "WalletRechargeRequest"
    mandatoryContents = [
        "Subscription-Id"
        "TopUp-RequestedAction"
        "Charging-Start-Timestamp"
        "Topup-Amount"
        "Topup-Voucher-Number"
        "Topup-Voucher-Type"
        "Topup-Voucher-Id"
        "Topup-Voucher-Serial-Number"
        "Source-System-Id"
    ]
    optionalContents = [
        "Voucher-Recharge_Failed-Flag"
        "Voucher-Recharge_Failed_Date_Time"
        "Topup-Voucher-Balance-Validity-Start"
        "Topup-Voucher-Balance-Validity-Relative"
    ]
}

{
    name = "Voucher-Recharge_Failed-Flag"
    type = "Integer32"
    vendorId = 581
    avpCode = 50998
    ccsConcept = "voucherRechargeFailureFlag"
    includeIf = true
}

{
    name = "Voucher-Recharge_Failed-Date-Time"
    type = "Integer32"
    vendorId = 581
    avpCode = 50999
    ccsConcept = "voucherRechargeFailureDateTime"
    excludeWhenIn = "0"
}

{
    name = "TopUp-RequestedAction"
    type = "Enumerated"
    value = "4"
    avpCode = 436
}

{
     name = "Charging-Start-Timestamp"
     type = "Integer32"
     profileBlock = 18
     profileTag = 327999
     avpCode = 12000
}
```

```
{
     name = "Topup-Voucher-Number"
     avpCode = 12001
     ccsConcept = "voucherInfoVoucher"
     type = "Integer32"
}

{
     name = "Topup-Amount"
     avpCode = 12002
     type = "Grouped"
     mandatoryContents = [
         "Value-Digits"
        ]
     optionalContents = [
         "Exponent"
     ]
}

{
    name = "Value-Digits"
    avpCode = 12003
    ccsConcept = "voucherInfoValue"
    type = "Integer32"
}

{
    name = "Topup-Voucher-Type"
    avpCode = 12005
    ccsConcept = "voucherTypeName"
    type = "Unsigned64"
}

{
    name = "Topup-Voucher-Id"
    vendorId = 581
    avpCode = 50026
    ccsConcept = "voucherInfoVoucherId"
    type = "Integer32"
}

{

    name = "Topup-Voucher-Serial-Number"
    avpCode = 12008
    ccsConcept = "voucherInfoVoucherSerialNumber"
    type = "Unsigned64"
}

{
    name = "Topup-Voucher-Balance-Validity-Start"
    avpCode = 12009
    ccsConcept = "voucherInfoBalanceValidityStart"
    type = "Time"
}

{
    name = "Topup-Voucher-Balance-Validity-Relative"
    avpCode = 12010
    type = "Grouped"
    optionalContents = [
        "Topup-Voucher-Balance-Validity-Offset"
        "Topup-Voucher-Balance-Validity-Type
```

```
    ]
}

{

    name = "Topup-Voucher-Balance-Validity-Offset"
    avpCode = 12011
    ccsConcept = "voucherInfoBalanceValidityOffset"
    type = "Integer32"
}

{
    name = "Topup-Voucher-Balance-Validity-Type"
    avpCode = 12012
    ccsConcept = "voucherInfoBalanceValidityType"
    type = "Enumerated"
}

{
    name = "WalletRechargeResponse"
    mandatoryContents = [
        "Topup-Balance-Information"
        "Topup-Receipt-Number"
    ]
}

{
    name = "Topup-Balance-Information"
    vendorId = 581 # Intec
    avpCode = 50030
    type = "Grouped"
    mandatoryContents = [
        "Topup-Balance-Type-ID"
    ]
    optionalContents = [
        "Topup-Balance-Expire-Date"
        "Topup-Unit-Value"
    ]
}

{
    name = "Topup-Balance-Type-ID"
    avpCode = 50020
    vendorId = 581 # Intec
    ccsConcept = "walletInfoBalanceType"
    type = "Integer32"
}
{
    name = "Topup-Balance-Expire-Date"
    vendorId = 581 # Intec
    avpCode = 50032
    ccsConcept = "walletInfoBalanceExpiry"
    type = "Integer32"
}

{
    avpCode = 445
    vendorId = 581
    name = "Topup-Unit-Value"
    type = "GroupedUnitValue"
    ccsConcept = "walletInfoBalanceSystemValue"
    conversionScale =
    signInversion - true
```

```
    mandatoryContents = [
        "Topup-Value-Digits"
    ]
    optionalContents = [
        "Topup-Exponent"
    ]
}

{
    avpCode = 447
    name = "Topup-Value-Digits"
    type = "Integer64"
}

{
    avpCode = 449
    name = "Topup-Exponent"
    type = "Integer32"
    literal = "1"
}

{
    name = "Topup-Receipt-Number"
    avpCode = 50024
    type = "UTF8String"
}

{
    name = "CreateEDRRequest"
}

{
    name = "CreateEDRResponse"
}


{
    name = "BadPINRequest"
}

{
    name = "BadPINResponse"
}

{
    name = "Termination-Cause"
    avpCode = 295
    type = "Enumerated"
    ccsConcept = "terminationCause"
    conversion = [
        {
            esg = 8 # releasedNoFunds
            vendor = 6 # DIAMETER_AUTH_EXPIRED
        }
        {
            esg = 9 # disconnectedLegBNoFunds
            vendor = 6 # DIAMETER_AUTH_EXPIRED
        }

        {
            esg = 14 # callingPartyDisconnected
            vendor = 1 # DIAMETER_LOGOUT
        }
        {
```

```
                esg = 15 # calledPartyDisconnected
                vendor = 1 # DIAMETER_LOGOUT
            }
        ]
    }

]

ContextCopy = [
{
    contextItem = "scpActionSupervise"
    profileBlock = 17
    profileTag = 6356992 # Hex 0x00610000
    profileFormat = "INTEGER"
}
{
    contextItem = "scpActionDoNotSupervise"
    profileBlock = 17
    profileTag = 6356993 # Hex 0x00610001
    profileFormat = "INTEGER"
}
{
    contextItem = "scpActionRelease"
    profileBlock = 17
    profileTag = 6356994 # Hex 0x00610002
    profileFormat = "INTEGER"
}
{
    contextItem = "scpActionSendMessage"
    profileBlock = 17
    profileTag = 6356995 # Hex 0x00610003
    profileFormat = "INTEGER"
}
{
    contextItem = "scpActionPlayAnnouncement"
    profileBlock = 17
    profileTag = 6356996 # Hex 0x0061004
    profileFormat = "INTEGER"
}
{
    contextItem = "scpActionSuperviseWithoutControlling"
    profileBlock = 17
    profileTag = 6356997 # Hex 0x0061005
    profileFormat = "INTEGER"
}
{
    contextItem = "callState"
    profileBlock = 17
    profileTag = 6356998 # Hex 0x0061006
    profileFormat = "INTEGER"
}
{
    contextItem = "sendCount"
    profileBlock = 17
    profileTag = 6356999 # Hex 0x0061007
    profileFormat = "INTEGER"
}
{
    contextItem = "preCallAnnouncementId"
    profileBlock = 17
    profileTag = 6357000 # Hex 0x0061008
    profileFormat = "INTEGER"
}
```

```
        {
            contextItem = "preCallLowBalance"
            profileBlock = 17
            profileTag = 6357001 # Hex 0x0061009
            profileFormat = "INTEGER"
        }
        ]

        TimeIn = {
            profileBlock = 17
            profileTag = 6357002 # Hex 0x006100a, continuing from example ContextCopy
            values
            profileFormat = "TIME"
        }
        TimeOut = {
            profileBlock = 17
            profileTag = 6357003 # Hex 0x006100b
            profileFormat = "TIME"
        }

    }
    ] # End of DomainTypes array

    PeerSchemes = [
    {
        schemeName = "SchemeA"

        Peers = [
        {
            name = "host1"

            scheme = [ "scheme1", "schmeme2" ]

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

            peer_group = "host1"

            transport = "tcp"

            initiation = "connect"

            RemoteAddresses = [
                "192.168.1.10"
            ]

            remote_port = 3868

            netmask6Bits = 128

            netmaskBits = 32

            permittedInstances = 0

            reqSctpInboundStreams = 8
            reqSctpOutboundStreams = 8

            sctp_hbinterval = 1000

            watchdogPeriod = 30

            connectionTimeout = 30
```

```
            inBufferSize = 0
            outBufferSize = 0

        } # end of Peer host1

        {
            name = "host2"

            scheme = [ "scheme1", "schmeme2" ]

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

            peer_group = "host1"

            transport = "tcp"

            initiation = "connect"


            RemoteAddresses = [
                "192.168.1.11"
            ]

            remote_port = 3868

            netmask6Bits = 128

            netmaskBits = 32

            permittedInstances = 0

            reqSctpInboundStreams = 8
            reqSctpOutboundStreams = 8

            sctp_hbinterval = 1000

            watchdogPeriod = 30

            connectionTimeout = 30

            inBufferSize = 0
            outBufferSize = 0

        } # end of Peer host1

    } # End of Scheme A

    {
        schemeName = "SchemeB"

        Peers = [
        {
            name = "host1"

            scheme = [ "scheme1", "schmeme2" ]

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

            peer_group = "host1"
```

```
            transport = "tcp"

            initiation = "connect"

            RemoteAddresses = [
                "192.168.1.10"
            ]

            remote_port = 3868

            netmask6Bits = 128

            netmaskBits = 32

            permittedInstances = 0

            reqSctpInboundStreams = 8
            reqSctpOutboundStreams = 8

            sctp_hbinterval = 1000

            watchdogPeriod = 30

            connectionTimeout = 30

            inBufferSize = 0
            outBufferSize = 0

        } # end of Peer host1
    ]
    } # End of Scheme B
    ] # End of PeerSchemes section

    routes = [
        {
            realm = "FirstRealm"
            host = "host1.realm1.oracle.com"

            priority = 1
            round_robin = 0
            direct = true
        }
    ]

    NamedEventTypes = [
        {
            eventClass = "abc"
            eventName = "def"
            eventType = 123
            isDebit = true
        }
        {
            eventClass = "ghi"
            eventName = "jkl"
            eventType = 456
        }
    ]

    HostSpecificData = [
        {
            name = "ocpc.oracle.com"
```

```
            DCD = {
                Origin-Host = "ocpc.oracle.com"
                Origin-Realm = "ocpc.oracle.com"
            }

            routes = [
                {
                    realm = "myDomainA"
                    host = "host1.realm1.oracle.com"

                    priority = 1
                    round_robin = 0
                    direct = true
                }
            ]

        }

    ]

}
```

Statistics Logged by diameterBeClient

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.

Enabling Statistics

Follow these steps to enable statistics on an SCP after installing the database entries on the SMF.

Step	Action
1	On the **Table Replication** tab of the SMS Node Management screen, select the DCD replication entry: `SMS->SMF_STATISTICS_DEFN->SMF_STDEF_DCD` and drag it over to the allocated Replication Groups. Refer to *Configuring Table Replication* in the *SMS User's Guide* for details.
2	After creating the config file, you need to send a HUP to the smsStatsDaemon to force a reread of the database entries: `# fuser -s 1 /IN/service_packages/SMS/bin/smsStatsDaemon`

DCD Statistics

SMS statistics are logged with APPLICATION_ID = ‘DCD’ (application number 97)

The following statistics are defined:

- DIAMETER_MIN_LATENCY – Minimum Latency
- DIAMETER_MAX_LATENCY – Maximum Latency
- DIAMETER_AVERAGE_LATENCY – Average Latency
- DIAMETER_CC_TYPE_INITIAL – Initial Requests sent

- DIAMETER_CC_TYPE_UPDATE – Update Requests sent
- DIAMETER_CC_TYPE_TERMINATION – Termination Requests sent
- DIAMETER_CC_TYPE_EVENT – Event Requests sent
- DIAMETER_TIMEOUT – CCRs timed out
- DIAMETER_SUCCESS – CCAs received with success result code
- DIAMETER_FAILURE_3xxx – CCAs received, error code in range 3000 to 3999
- DIAMETER_FAILURE_4xxx – CCAs received, error code in range 4000 to 4999
- DIAMETER_FAILURE_5xxx – CCAs received, error code in range 5000 to 5999
- DIAMETER_FAILURE_1xxx – CCAs received, error code in range 1000 to 1999
- DIAMETER_FAILURE_UNKNOWN – CCAs received, error code in undefined range
- RAA_GENERATED_SENT – RAA generated from stored RAR, send to server
- RAA_NO_OUTSTANDING_RAR – Rejecting received RAA, no record of RAR
- RAA_RECEIVED – RAA received from DCA
- RAA_SENT – RAA send to Diameter server
- RAA_SENT_WITH_ERROR – RAA sent to Diameter server, with error indication
- RAA_TIMEOUT_ALREADY_CLEARED – RAA received, but timeout already cleared
- RAA_UNKNOWN_SESSION_ID – RAA rejecting received RAA, unknown session ID
- RAR_RECEIVED – RAR received from diameter server
- RAR_SEND_FAIL – RAR failed to send RAR to DCA, no dialog etc.
- RAR_SENT – RAR sent to DCA
- RAR_TIMEOUT_RAA_SENT – RAR send timeout, and RAA reject sent to server
- TERM_REQUEST_BEFORE_RAA – Termination request before RAA received
- RAR_UNKNOWN_SESSION_ID – RAR received with unknown session ID
- RAR_NOT_ENABLED_RESPONSE – RAR received gets configured resultcode

For all statistics, the Destination-Realm or Host ID involved is put into SMF_STATISTICS.DETAIL.

Reports

The following reports are available:

- DCD System Stats
- DCD System Stats by Realm/Host

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

Example Report

Here is an example DCD System Stats report.

```
DCD Statistics Listing
==============================
Start Date: 15 October 2007
Finish Date: 30 October 2007
Report Type: Totals

24 October 2007, 21:20:12

Node Name         Statistics ID                                      Totals
------------------- ------------------------------------------------ ----------
mtv-tst-scp10     DIAMETER_FAILURE_UNKNOWN                                3
mtv-tst-scp10     DIAMETER_SUCCESS                                      319
mtv-tst-scp10     DIAMETER_FAILURE_5xxx                                  14
mtv-tst-scp10     DIAMETER_CC_TYPE_INITIAL                              214
mtv-tst-scp10     DIAMETER_FAILURE_1xxx                                   2
mtv-tst-scp10     DIAMETER_TIMEOUT                                       63
```

```
mtv-tst-scp10     DIAMETER_CC_TYPE_UPDATE                                185
mtv-tst-scp10     DIAMETER_FAILURE_3xxx                                    8
mtv-tst-scp10     DIAMETER_CC_TYPE_TERMINATION                            86
mtv-tst-scp10     DIAMETER_CC_TYPE_EVENT                                  14
mtv-tst-scp10     DIAMETER_FAILURE_4xxx                                   39

Completed
```

DCD EDRs

EDR Generation

EDRs are generated and processed by the slee_acs on the SLC and uploaded at regular intervals to the SMS using the cmnPushFiles process.

Diameter Charging Driver (DCD) tags are appended to the EDRs generated by the Advanced Control Services application. See *Event Detail Record Reference Guide* for the full list and descriptions.

DCD EDR Tags

Here are the EDR tags produced by DCD.

`DIA_RC (result code)`

Description:	Number indicating diameter result-code received in CCA message.
Format:	Integer
Concept:	Result-Code
Notes:	
Example:	`DIA_RC=2001`

`DIA_REQ (current session message number)`

Description:	Sequential number, indicating message within the current session.
Format:	Integer
Concept:	CC-Request-Number
Notes:	For Diameter event based messages, this will always be 0, and hence not recorded.
Example:	`DIA_REQ=1`

`DIA_SID (session id)`

Description:	This is a unique value identifying the Diameter session.
Format:	Of the form: *DiameterIdentity*;*time*;*SLEE_CallID* Where: • *DiameterIdentity* is that of the SLC (that is, the Origin-Host used in the CCR message) • *time* is the time of the first request (expressed as the number of seconds since the Unix epoch time) • *SLEE_CallID* is a unique call identifier used by the SLEE processes to track each active session
Concept:	Session-ID
Notes:	The values for *time* and *SLEE_CallID* are in decimal format, but they are actually sent out in hexadecimal format.

Example: DIA_SID=scp1.oracle.com;47A228C3;15459A

DIA_TIME (time ccr sent)

Description: The time the CCR was sent, in hundredths of second
Format: Date - "YYYY-MM-DD-HH-MM-SSSS"
Concept: Session-ID
Notes:
Example: DIA_TIME=2008-03-27-20-41-3831

Custom Tag Names

The cdrTag configuration parameter allows for an EDR to have tag names that are customer defined.

Example EDRs

Here are some example EDRs generated by DCD.

Refer to *ACS EDR Tags* for the non-DCD tags.

Example 1

Whole EDR for an InitialTimeReservation and ConfirmTimeReservation:

```
EDR:
'VOICE_MO|CID=285222|OA=0|OTI=0|CUST=1|SN=0777666444|TN=0777666444|CGN=8888887|CLI=8
888887|SK=1|TCS=20080327204138|TCE=20080327204241|LPN=|LAC=|CS=4|CPC=10|CC=|CPNI=0|P
CNA=|TPNI=0|PTNA=|CGNA=|TGNA=|TFN=ST-2,SDTN-21,uatb-3,PB-22,END-
14|LGID=0|CPN=uatbWcseBrch|CAET=3|CCET=60.0|CA=60777666555|RELC=17|OCPI=|CPNN=3|CGNN
=3|CPPI=1|NOAT=1|CBAT=0|FATS=0|CCTS=20080327204138|HTS=20080327204138|AIDL=|DIA_SID=
nzwn-test03-z2;47ec0682;45a26|DIA_REQ=0|DIA_RC=2001|DIA_TIME=2008-03-27-20-41-
3831|DIA_SID=nzwn-test03-z2;47ec0682;45a26|DIA_REQ=1|DIA_RC=2001|DIA_TIME=2008-03-
27-20-41-3847|FCA=60777666555|WALR=86'
```

Note the DCD part of the EDR. All four tags are present twice:

- The Initial Time Reservation
  DIA_SID=nzwn-test03-z2;47ec0682;45a26|DIA_REQ=0|DIA_RC=2001|DIA_TIME=2008-03-27-20-41-3831|
- The Termination Time Reservation
  DIA_SID=nzwn-test03-z2;47ec0682;45a26|DIA_REQ=1|DIA_RC=2001|DIA_TIME=2008-03-27-20-41-3847|

Example 2

Whole EDR for DirectNamedEvent:

```
EDR:
'VOICE_MO|CID=287224|OA=0|OTI=0|CUST=1|SN=0777666444|TN=|CGN=8888887|CLI=8888887|SK=
1|TCS=20080327231115|TCE=0|LPN=|LAC=|CS=1|CPC=10|CC=|CPNI=0|PCNA=|TPNI=0|PTNA=|CGNA=
|TGNA=|TFN=ST-1,bevt-2,END-
3|LGID=0|CPN=DirectDebit|CAET=0|CCET=0.0|CA=|RELC=31|OCPI=|CPNN=3|CGNN=3|CPPI=1|NOAT
=0|CBAT=0|FATS=0|CCTS=0|HTS=0|AIDL=|DIA_SID=nzwn-test03-
z2;47ec2993;461f8|DIA_RC=2001|DIA_TIME=2008-03-27-23-11-1577'
```

Note in the DCD part of the EDR that event based EDRs only need to receive DIA_SID, DIA_RC and DIA_TIME, that is, no DIA_REQ:

```
DIA_SID=nzwn-test03-z2;47ec2993;461f8|DIA_RC=2001|DIA_TIME=2008-03-27-23-11-1577'
```

Example 3

For cdrTag for MMM_TAG and ZZZ_TAG, the following would be an example of what the resulting EDR would look like.

```
CCS_BE|CID=205383|OA=0|OTI=0|CUST=1|SN=1130|TN=|CGN=0212994768|CLI=0212994768|SK=3|T
CS=20091117192600|LPN=|LAC=|CS=1|CPC=10|CC=|CPNI=0|PCNA=|TPNI=0|PTNA=|CGNA=|TGNA=|TF
N=ST-1,CCDR-8,CCDR-12,CCDR-14,bevt-2,DISC-3,END-7|LGID=0|CPN=ST-BE-
END|OCPI=|CPNN=3|CGNN=3|CPPI=1|NOAT=0|CBAT=0|FATS=0|CCTS=0|HTS=0|AIDL=|AAA_TAG=11111
|CMX_EC=CR96791|CMX_EN=BasicTest|DIA_SID=eng-host06-
z6.usp.co.nz;4b02f8c8;32247|DIA_RC=2001|DIA_TIME=2009-11-17-19-26-
0062|MMM_TAG=55555|ZZZ_TAG=777
```

Chapter 6

About Installation and Removal

Overview

Introduction

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

In this Chapter

This chapter contains the following topics.

Installation and Removal Overview

Introduction

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

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

DCD Packages

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

- SMS:
  - dcdSms
- SLC:
  - dcdScp

Checking the Installation

Introduction

Refer to these check lists to ensure the package has been installed correctly.

DCD SLC Directories and Files

The DCD installation on the SLC creates the following directories:

- **/IN/service_packages/DCD/bin**
- **/IN/service_packages/DCD/etc**

- **/IN/service_packages/DCD/lib**
- **/IN/service_packages/DCD/tmp**

The DCD installation installs the following binaries and interfaces:

- **/IN/services_packages/DCD/bin/diameterBeClient**

The DCD installation installs the following example configuration file:

- **/IN/services_packages/DCD/etc/eserv.config.dcd.example**

The DCD installation installs the following shared library:

- **/IN/services_packages/DCD/lib/diamActions.so**

DCD SMS Directories

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

The DCD installation on the SMS creates the following directories:

- **/IN/service_packages/DCD/db**
- **/IN/service_packages/DCD/lib**