Copyright

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

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

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

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

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

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

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

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

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

Scope
The scope of this document includes all the information required to install, configure and administer the VWS application.

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

Prerequisites
Although there are no prerequisites for using this guide, familiarity with the target platform would be an advantage.

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 and complete loss of service.

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:

- SLEE Technical Guide
- SMS Technical Guide
- SMS User's Guide
- NCC Event Detail Record Reference Guide

If this Voucher & Wallet Server is installed with Charging Control Services, these documents are also related to this document:

- CCS Alarms Reference Guide
- CCS Technical Guide
- CCS User's Guide
- NCC Feature Nodes Reference Guide
Document Conventions

Typographical Conventions

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

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

Specialized terms and acronyms are defined in the **Glossary** at the end of this guide.
Chapter 1

System Overview

Overview

Introduction

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

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

In this chapter

This chapter contains the following topics.

- Introduction to VWS 1
- Wallets, Balances and Buckets 4
- Request handling 7
- Data Management 11
- Synchronization 14
- Wallet and Bucket Expiry 17
- BE States 20
- EDR Processing 22
- MFile Updates 24
- Statistics 26

Introduction to VWS

Introduction

The Voucher & Wallet Server (VWS) provides high-performance, real-time charging and subscriber account management functions.

The VWS solution maintains voucher, wallet and reservation details in the E2BE database on the Voucher and Wallet Server. It enables call processing applications to bill mobile customers.

Billing information is kept logically separate from call processing on the SLCs, allowing it to be used by multiple clients.

Functions

The role of the VWS is to manage all the billing/charging information associated with call processing.

The VWS provides:

- Subscriber account management
- Management of multiple wallets and balance types
- Real-time rating of services
- Structure for tariffing on transactions
- Reservation, debit and credit requests
- Voucher management, including voucher query and redemption requests
- Failover and machine redundancy
- EDRs and EDR archiving

**Main components diagram**

This diagram shows the main components of VWS. They mostly run as separate processes on the SLEE, as illustrated below. Their operation needs to be coordinated, as the state of the entire VWS affects their behavior.

**Main components**

This table describes the main components in VWS.

<table>
<thead>
<tr>
<th>Process</th>
<th>Role</th>
<th>Further information</th>
</tr>
</thead>
<tbody>
<tr>
<td>BeClient</td>
<td>The BeClient is a SLEE interface that runs on the SLC and handles any process that uses the libBeClientIF library to connect to the beServer. The main BeClient is the BeClient provided by VWS for the SLC. (Other applications may provide other processes to handle other activities such as ccsBeOrb, which handles interaction between the SMS UI and the Voucher and Wallet Server nodes.)</td>
<td>BeClient (on page 44)</td>
</tr>
<tr>
<td>beServer</td>
<td>Handles connections from client processes (including BeClient processes) and controls routing to beVWARS processes. You can run more than one beServer process to improve performance on Sun CMT hardware.</td>
<td>beServer (on page 58)</td>
</tr>
</tbody>
</table>
| beVWARS | beVWARS is the core of the VWS. More than one beVWARS will usually be running on a VWS. It:  
- Reads and caches wallet and voucher | beVWARS (on page 85) |
information from the E2BE database
- Manages all queries, reservations and updates against wallets
- Manages all queries, redemptions and state changes for vouchers
- Writes files used to synchronize data
- Writes EDRs.

beVWARS plug-in handlers
Perform business-case specific operations on wallets and vouchers. Some plug-in handlers are provided by VWS, but other applications can extend VWS logic by providing additional plug-ins.

Examples: CCS beVWARS plug-in handlers manage monthly:
- Spend accumulation and upgrade
- Account activation

beSync
Synchronizes data between the Voucher and Wallet Servers in a VWS pair. You can run more than one beSync process to improve performance on Sun CMT hardware.

E2BE database
The databases on the VWSs. They hold a subset of the data from the SMF database on SMS.

### Billing Interfaces

VWS supports external interfaces to bill for third-party services, such as:
- Calling card services
- Data charging services
- SMS charging services
- Universal Parlay Gateway

### VWS Domains

Each VWS domain is made up of a pair of Voucher and Wallet Server. For more information about Voucher and Wallet Server pairs, see Data redundancy (on page 11).

Domains can provide a full set of functions, or may be configured to offer a specific set of functions. Domains are configured in the Service Management screen in CCS. For more information about configuring domains, see CCS User's Guide.

### About improving performance

If the VWS server will be processing high volumes of traffic, you may be able to improve performance by configuring the SLEE to run multiple instances of the beServer and beSync processes. Running multiple instances of these processes enables the VWS server to load-share:
- beServer client connections over multiple process spaces
- beSync synchronization connections over multiple process spaces

For information on configuring beServer, see beServer (on page 58). For information on configuring beSync, see beSync (on page 69).
Note: In the diagrams in the following sections in this guide that show the beServer and beSync processes, only one instance of each is shown.

WALLETS, BALANCES AND BUCKETS

WALLETS

Each subscriber account is linked to one or two wallets.

A wallet is a group of balances owned by the subscriber and available to pay for prepaid services offered by the platform.

Example: A subscriber could have a “General Cash” balance and a “Free Notification” balance in their wallet. Each balance has its own expiry date, which means that any value left in the balance after this date will be removed.

WALLET STATES

The wallet has a state which:

- Indicates what phase of the life cycle the associated subscriber account currently is in
- Determines whether the subscriber can use his or her services

This table describes the available wallet states.

<table>
<thead>
<tr>
<th>State</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pre-use</td>
<td>The first state after the subscriber account/wallet is created.</td>
</tr>
<tr>
<td>Active</td>
<td>The state during which normal wallet operations are handled. All services can be used.</td>
</tr>
<tr>
<td></td>
<td>When a subscriber uses their account for the first time (via any paid service), the wallet moves from the Pre-use state into the Active state. The initial expiry dates for the balances and the wallet are set when the wallet is activated.</td>
</tr>
<tr>
<td>Dormant</td>
<td>If the subscriber does not use any services for a configurable period of time, the account/wallet is put in the Dormant state. The Dormant state is useful for reporting inactive subscribers. All services are still accessible.</td>
</tr>
<tr>
<td>Frozen</td>
<td>If fraudulent activity is detected on the subscriber account, the subscriber account/wallet is changed into the Frozen state. All services are disabled until manual intervention by an operator.</td>
</tr>
<tr>
<td>Suspended</td>
<td>The suspended state can be used by the operator to temporarily disable a subscriber's account.</td>
</tr>
<tr>
<td>Terminated</td>
<td>When the wallet's expiry date is reached, the subscriber account/wallet is moved into the terminated state. All services are disabled.</td>
</tr>
</tbody>
</table>

Note: Wallets may also be affected by expiry dates. For more information about wallet and bucket expiry, see Wallet and Bucket Expiry (on page 17).
Wallet lifecycle

This diagram shows the states in a standard life cycle of a wallet.

![Wallet lifecycle diagram]

Wallet life cycle plans

A wallet life cycle plan comprises a set of wallet life cycle periods. The plan may be associated with a subscriber's wallet through the subscriber's product type.

The plan is used to extend the existing states of the wallet with customizable sub-states called periods. The beginning of each period can be configured as an offset of days before or after the wallet expiration date. A period ends when the next one, if defined, starts or when the wallet expires.

For each period in the wallet life cycle you can define the features that will be available. This includes:

- Session charges
- Available named event operations
- Enabled general charges and recharges

For information on configuring wallet life cycle plans, see the Wallet Management section in *CCS User's Guide*.

VWS associations

When a wallet is created, it is created on a specific VWS Voucher and Wallet Server pair. This Voucher and Wallet Server pair handles all the updates and information queries for that wallet.

Processes which send a wallet request will usually check to which VWS to send the request before the request is sent. If the wallet request is sent to the wrong VWS, the VWS will return an error.

Migrating wallets

If the Voucher and Wallet Server (VWS) pair is storing CCS wallets, the wallets can be migrated from one VWS to another using the UBE Account Balancing tab. For more information about this tab, see *CCS User's Guide*. For more information about how CCS migrates wallets between VWSs, see *CCS Technical Guide*.

Balances

Balances record a value in a wallet by collecting buckets into a group. Each bucket records a specific value with an optional expiry date. There are two main types of balances:
- Chargeable balances which record value that can be used for services
- Internal balances which are used for internal values and triggers within the system

**Wallets, balances and buckets relationship**

This diagram shows how buckets are collected into a balance value, and balances are connected to a wallet.

**Wallets**

Balance Types

- General Cash
- Promotion Cash
- SMS

Balances (expiry)

- $12 12/10/04
- $25 20/11/04
- $2 23/10/04
- $1 09/11/04
- 60 15/10/04

**Note:** Internal balances have the same structure.

**Wallet and bucket events**

When beVWARS receives a request which involves a wallet, it will load the wallet and all the wallet's buckets. This includes loading a wallet for a query, update or charging operation. When the wallet is loaded, wallet load event plug-ins are triggered. These event plug-ins will take any required actions on the wallets and buckets as necessary.

**Background processing**

In normal processing, events are triggered only when a subscriber or customer care representative interacts with the wallet. Some events (such as expiries and periodic charges) should be triggered regardless of whether the wallet has been used by a subscriber or customer care representative. In order to process these events, beGroveller sends lists of wallet IDs to beVWARS for processing. This processing triggers any events which are due to occur in the same way a normal interaction would, except wallet events triggered from beGroveller lists do not trigger any message handlers.

For more information about how wallets and buckets are expired, see *Wallet and Bucket Expiry* (on page 17).
Request handling

Reservations and billing diagram

This diagram shows how VWS handles requests.

beVWARS plugins

beVWARS plug-ins can trigger on any event which requires reading data about a wallet or voucher (including maintenance through SMS UI, call charges, or named events). Because the plug-ins run before the original request is processed, any action such as expiring a bucket or wallet is executed before any charges are applied. This means expired wallets or buckets are never available, even if they still exist in the database.

Applications which are using the VWS for charging or voucher redemption usually provide beVWARS plug-ins to handle the specific application logic required by the application. VWS provides these plug-ins as standard:

- beVWARSExpiry (on page 105)
- beVWARSMergeBuckets (on page 107)
- libbeEventFactory (on page 115)

Request processing

This table describes how VWS handles requests from service applications.
Note: For information about how CCS handles charging for calls or SMS and the CCS plug-ins mentioned in the process, see CCS Technical Guide.

<table>
<thead>
<tr>
<th>Stage</th>
<th>Description</th>
</tr>
</thead>
</table>
| 1     | Requesting process sends a request to the BeClient process (usually BeClient on the SLC, but also PiBeClient and ccsBeOrb on the SMS).  
Note: Other applications and specific functions may use other BeClient processes, but the ones mentioned are the most common. |
| 2     | BeClient process checks for plug-ins to handle this message. Plug-ins are specified in the plugins (on page 50) parameter in eserv.config.  
Example: libBeClientIF (on page 114) will usually be triggered in addition to any application-specific plug-ins such as libccsClientPlugins which applies CCS logic. |
| 3     | If the message has a BE id of 0, libclientBcast will send the request to all VWS Voucher and Wallet Servers to locate the Voucher and Wallet Server which holds the details which are relevant to the request.  
If the message has a BE id other than 0, BeClient will send the request to the VWS pair with that id.  
Note: The BE id of a VWS is set by the serverId (on page 36) parameter.  
Example: If the request is a voucher redeem, the libclientBcast library will cause the BeClient to send a request to all Voucher and Wallet Server pairs to locate the Voucher and Wallet Server pair which holds the data for the voucher which is about to be redeemed. |
| 4     | beServer receives the request from the BeClient process via FOX over TCP/IP. It determines the message type and checks whether there are any handlers for this message. Handlers are configured in the handlers (on page 62) parameter in eserv.config. |
| 5     | beServer creates a context to store information for the request. The information includes:  
- The original request  
- The BeClient  
- Any message handler that is handling the request  
- The state the message handler is in  
- The beVWARS process which will handle the request |
| 6     | beServer checks for and executes routing plug-ins configured in the messageRoutingPlugins (on page 63) parameter in eserv.config.  
Note: If this message is part of a sequence (but not the first), beServer will send the message to the same beVWARS as the other parts of the sequence. |
| 7     | beServer forwards the message across the SLEE to the correct beVWARS. |
| 8     | beVWARS determines whether there is a message handler for this message type. Message handlers are configured in the handlers (on page 88) parameter in eserv.config.  
Note: If there is no message handler for this type, beVWARS will log an error. |
| 9     | When the initial message handler is triggered, it will query either the wallet or the voucher cache. If the cache does not contain the details or the details in the cache are stale, beVWARS reads the details from the database. |
| 10    | Depending on the message type, different event and message plug-ins will be triggered. Plug-ins (including event handlers) are configured in the plugins (on page 90) parameter in eserv.config. |
Example: If a wallet is interacted with, beVWARSExpire will check the expiry dates on all buckets in the wallet, and will expire any buckets which are overdue. For more information about expiry handling, see Wallet and Bucket Expiry (on page 17).

Note: At least one plug-in must be provided to process requests from an application which is using VWS for charging. For more information about the specific plug-ins which are triggered, what order they are triggered in, and what they do, see the technical guide for that application.

For updates and reservations (but not queries), beVWARS starts the synchronization process by writing the change to the sync files. Synchronization enables the VWS to replay operations in the event of a failure and maintain redundancy in the event of a failure. For more information about how the synchronization process works, see Data synchronization.

For updates only, beVWARS also flushes the data. It:

- Updates the wallet cache from the database
- Updates the database (this update will be held in a queue until the next flush of COMMITs to the E2BE database)
- Writes the changes to the EDR file (these updates are also queued for bulk writing)

Note: The EDRs will be written by the beVWARS which executed the update. This avoids duplicate EDRs being written in the event of a failover.

For more information about how beVWARS writes data, see Queuing and flushing updates (on page 12).

For more information about EDRs, see EDR Processing (on page 22).

beVWARS on the primary VWS sends the acknowledgment back to BeClient via beServer.

BeClient passes the acknowledgment back to the requesting process.

Wallet and voucher caches

beVWARS maintains a wallet cache and a voucher cache to store up to date information about the wallets and vouchers it maintains. beVWARS updates the cache record for a wallet or voucher whenever one of the following occurs:

- Wallet or voucher is queried
- Wallet reservation or update is received
- Voucher is redeemed

A wallet or voucher record is removed from the cache if the record for it expires before a new request for that record is retrieved.

Supported requests

This table describes the types of messages VWS supports.

<table>
<thead>
<tr>
<th>Message Type</th>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial Reservation</td>
<td>IR</td>
<td>Reserve a charge amount.</td>
</tr>
<tr>
<td>Subsequent Reservation</td>
<td>SR</td>
<td>Reserve another charge amount.</td>
</tr>
<tr>
<td>Commit Reservation</td>
<td>CR</td>
<td>Apply reserved charge.</td>
</tr>
<tr>
<td>Revoke Reservation</td>
<td>RR</td>
<td>Abandon a reservation.</td>
</tr>
<tr>
<td>Named Event</td>
<td>NE</td>
<td>Attempt to charge a named event.</td>
</tr>
</tbody>
</table>
### Named Event Rate
- **NER**
  - Used by Named Event feature node.

### Direct Amount
- **DA**
  - OSA CHAM (Charging and Account Management) named event.

### Initial Amount Reservation
- **IARR**
  - OSA CHAM initial reservation.

### Subsequent Amount Reservation
- **SARR**
  - OSA CHAM subsequent reservation.

### Commit Amount Reservation
- **CARR**
  - OSA CHAM commit reservation.

### Revoke Amount Reservation
- **RARR**
  - OSA CHAM revoke reservation.

### Unit Second Rate
- **USR**
  - OSA CHAM tariffed rate.

### Apply Tariffed Charge
- **ATC**
  - Used by DUCR feature node.

### Initial Named Event Reservation
- **INER**
  - Attempt to reserve a named event.

### Subsequent Named Event Reservation
- **SNER**
  - Named event reservation which follows on from an Initial Events Reservation (INER).

### Confirm Named Event Reservation
- **CNER**
  - Apply reserved named event.

### Revoke Named Event Reservation
- **RNER**
  - Abandon a named event reservation.

### Voucher Information
- **VI**
  - Query a voucher.

### Voucher Update
- **VU**
  - Update voucher details.

### Voucher Redeem
- **VR**
  - Reserves a voucher.

### Commit Voucher Redeem
- **CVR**
  - Wallet changed successfully, redeem voucher.

### Revoke Voucher Redeem
- **RVR**
  - Abandon a voucher reservation.

### Voucher Redeem Wallet
- **VRW**
  - Tells BeClient to start a Voucher Redemption.

### Voucher Type Recharge
- **VTR**
  - Recharge a wallet using a voucher type name.

### Voucher Type Recharge Confirm
- **VTRC**
  - Perform product type swap.

### Wallet General Recharge
- **WGR**
  - Recharge wallet and buckets.

### Wallet Update
- **WU**
  - Update wallet details (not buckets/balances).

### Wallet Create
- **WC**
  - Create a new wallet.

### Wallet Delete
- **WD**
  - Delete an existing wallet.

### Wallet Information
- **WI**
  - Query a wallet and its buckets/balances.

### Bad PIN
- **BPIN**
  - Increase the Bad PIN balance.

### Reload the MFile
- **LDMF**
  - Reload an updated MFile.

---

### Wallet life cycle period checks

Before processing a request, the request handler will check that the corresponding feature in the current wallet life cycle plan period is enabled. If the feature is disabled for the subscriber’s wallet, then the request will fail.

This table lists the request handler and the feature it checks for when processing requests.

<table>
<thead>
<tr>
<th>Request Handler</th>
<th>Wallet Life Cycle Period Features Checked</th>
</tr>
</thead>
<tbody>
<tr>
<td>IR</td>
<td>Session Charge</td>
</tr>
<tr>
<td>IARR</td>
<td>Session Charge</td>
</tr>
<tr>
<td>NE</td>
<td>General Charge and First Named Event Class if the amount is greater than or equal to 0 (debit) General Recharge and First Named Event Class if the amount is less than 0 (credit)</td>
</tr>
<tr>
<td>Request Handler</td>
<td>Wallet Life Cycle Period Features Checked</td>
</tr>
<tr>
<td>-----------------</td>
<td>------------------------------------------</td>
</tr>
<tr>
<td>INER</td>
<td>Session Charge and First Named Event Class</td>
</tr>
<tr>
<td>ATC</td>
<td>General Charge if the amount is greater than or equal to 0 (debit) General Recharge if the amount is less than 0 (credit)</td>
</tr>
<tr>
<td>DA</td>
<td>General Charge if the amount is greater than or equal to 0 (debit) General Recharge if the amount is less than 0 (credit)</td>
</tr>
<tr>
<td>WGR</td>
<td>General Recharge</td>
</tr>
</tbody>
</table>

### Merging wallets processes

This table describes the main components involved in merging wallets.

<table>
<thead>
<tr>
<th>Process</th>
<th>Role</th>
<th>Further information</th>
</tr>
</thead>
<tbody>
<tr>
<td>beServer</td>
<td>Handles connections from client processes (including BeClient processes) and controls routing to beVWARS processes.</td>
<td>beServer (on page 58)</td>
</tr>
<tr>
<td>beVWARS</td>
<td>beVWARS is the core of the VWS. More than one beVWARS will usually be running on a VWS. It:</td>
<td>beVWARS (on page 85)</td>
</tr>
<tr>
<td></td>
<td>- Reads and caches wallet and voucher information from the E2BE database</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Manages all queries, reservations and updates against wallets</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Manages all queries, redemptions and state changes for vouchers</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Writes EDRs.</td>
<td></td>
</tr>
<tr>
<td>beVWARSMergeBuckets</td>
<td>This beVWARS plug-in merges buckets in the same balance when there are too many buckets in the wallet.</td>
<td>beVWARSMergeBuckets (on page 107)</td>
</tr>
<tr>
<td>E2BE database</td>
<td>The databases on the VWSs. They hold a subset of the data from the SMF.</td>
<td>&quot;beVWARS plug-ins&quot; (see on page 7)</td>
</tr>
</tbody>
</table>

### Data Management

#### Data redundancy

In a redundant configuration there are two Voucher and Wallet Servers in each VWS domain: a primary and a secondary. The primary is the node with:

- 'true' specified in the amPrimary (on page 35) parameter
- The highest node number

In normal conditions, the primary VWS performs all subscriber account, wallet and balance actions for the pair. The secondary VWS maintains a duplicate set of data.

If a single Voucher and Wallet Server in a pair is down, the system will work as normal. When the other peer comes back up:
1. It will resynchronize with the uninterrupted peer without prompting
2. Service will continue as if nothing happened

If the network link between BeClients and beServers, or between peer servers in a redundant pair, is disconnected, those BeClients that can see at least one member of a VWS pair should be able to keep running. When connectivity is restored, changes made to records held on the peers are resynchronized.

If the primary VWS fails, the secondary VWS performs the functions of a primary VWS until the primary VWS becomes available again.

For more information about failover and recovery, see BE States (on page 20).

**BeClients and connection failure**

If a BeClient process cannot connect to the beServer on the primary VWS, it will retransmit any outstanding messages to the secondary VWS for processing. Subsequent messages will go to the secondary VWS, until the primary VWS recovers. When the primary VWS recovers, BeClient sends new transactions to the primary VWS.

This prevents call crossover conditions, where the beginning of a call could start on one VWS and end on another.

**Throttling**

If one beVWARS process is throttling, the beServer will stop accepting any new requests on the client sockets.

**Database update consistency**

The E2BE database seldom reflects the complete state of the running system, because updates are almost always pending. To maintain a single consistent view of the state of records in the database, all wallet or voucher accesses are executed through beVWARS. beVWARS is responsible for all updates to resynchronized database fields.

Because beVWARS sends updates to beSync before confirmation of the update on the local VWS has happened, updates may be applied to the remote VWS and not the local VWS. If the local VWS fails before updates have been applied, then the updates are retrieved and applied during resynchronization.

**beVWARS data updates**

Each beVWARS performs the following data update tasks:

- Buffering database updates and EDRs (for performance reasons)
- Applying database updates to the database in the order they were produced
- Writing EDR data to flat files
- Writing updates to sync files
- Informing beSync there is a sync file to process

**Queuing and flushing updates**

Updates fall into one of these categories:

- Bucket creations, updates or deletions
- Balance creations, updates or deletions
- Wallet creations, updates or deletions
- Voucher creations, updates or deletions

A buffer is maintained for each of these categories to store updates for later binding to the database via a bulk bind operation. Each buffer has a configurable maximum size.
beVWARS also maintains an EDR cache to queue EDR details for later writing to disk.

Each time beVWARS processes a transaction, it checks the following criteria to determine if it should perform a flush:

- One of the buffers is full
- The configurable buffer/cache flush period has been reached
- The writer subsystem is told to flush and commit (on shutdown, for example)

If a flush is triggered, all buffers are written to the E2BE database and the EDR cache is written to the EDR files.

### Flush process

This describes the stages involved in a flush operation.

<table>
<thead>
<tr>
<th>Stage</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>beVWARS writes all EDR records in the cache to a new file.</td>
</tr>
<tr>
<td>2</td>
<td>beVWARS records the EDR file name in the E2BE database (BE_CDR_FILE).</td>
</tr>
</tbody>
</table>
| 3     | Update the BE_VWARS_SEQ_NUM table entry for the current beVWARS process. Sets:  
|       |   - ‘last local sequence number’ to the sequence number of the last update in the buffers, and  
|       |   - ‘remote sequence number’ to the last value sent from the remote beSync). |
| 4     | Bulk bind and execute the database statement associated with each database buffer. The buffers are flushed in this order:  
|       |   - BE_WALLET, BE_BALANCE, BE_BUCKET, then BE_VOUCHER Inserts  
|       |   - BE_WALLET, BE_BALANCE, BE_BUCKET, then BE_VOUCHER Updates  
|       |   - BE_BUCKET, BE_BALANCE, BE_WALLET, then BE_VOUCHER Deletes |
| 5     | Commit the changes to the E2BE database. |
| 6     | Generate and send a COMMIT message to the beSync process containing the last local and remote sequence numbers. |
| 7     | Update the last committed local update sequence number. |
| 8     | Perform a wallet cache flush. This frees up space in the wallet cache by releasing any entries which were protected until a flush applied the transaction. |

### Changing number of beVWARS

Because of the method used to keep the VWS pairs synchronised, you must run the same number of beVWARS on both Voucher and Wallet Server nodes.

Follow these steps to change the number of beVWARS to run on the VWS.

**Note:** If you reduce the number of beVWARS on a pair, any transactional updates that are incomplete when the change is made will be lost.

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Set the number of beVWARS interface instances to run by editing the INTERFACE line for beVWARS in SLEE.cfg for each VWS in the pair.</td>
</tr>
<tr>
<td>2</td>
<td>Stop the SLEE on both VWSs in the pair. The VWS will move to the disabled state.</td>
</tr>
<tr>
<td>3</td>
<td>Start the SLEE on both VWSs in the pair.</td>
</tr>
</tbody>
</table>
Chapter 1

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>The SLEE will start the number of beVWARS instances specified in the updated SLEE.cfg file.</td>
</tr>
<tr>
<td></td>
<td>The VWS will move to a running state.</td>
</tr>
</tbody>
</table>

For more information about configuring SLEE interfaces, stopping the SLEE, and starting the SLEE, see SLEE Technical Guide.

Synchronization

Data synchronization

Synchronization is used to ensure database updates and EDRs are not lost. This is achieved by beVWARS recording a stream of updates to flat files, so the updates can:

- Be replayed in the event of a failure (for example, a hardware fault, power failure, software failure)
- Provide a persistent stream of updates to be delivered to the secondary VWS for application to a remote E2BE database in the interests of redundancy and failover

It is possible for the remote beSync to drop the connection and later connect and request older updates at any point. The sync files are only removed when both VWSs have committed the updates.

The current position in the transaction stream for each VWS within a pair is recorded as a pair of sequence numbers. These numbers record the last locally sourced update and the last remotely sourced update received and written to the database.

Sequence numbers are managed for each beVWARS, so lost updates are visible as gaps in the sequence numbers for the beVWARS handling that traffic.

Synchronization between two VWSs in a VWS domain can be viewed as two continuous streams containing all reservations, updates, and deletions made on one VWS to the other. The two streams are connections from:

1. Primary beSync source to secondary beSync sink.
2. Secondary beSync sink to primary beSync source.

The two stream connections between the beSync processes on the primary and secondary VWS nodes are maintained using the internal port that you specify when you configure the VWS domain. If you are running more than one instance of beSync on the VWS nodes, then two stream connections will be maintained for each instance. The internal port number is incremented by 1 (one) for each additional instance of beSync.

For information on configuring VWS domains, see the section on Service Management in CCS User's Guide.

Example

If you are running two instances of beSync and the internal port number is 1500 then the connections between:

- beSync0 on VWS1 and beSync0 on VWS2 use port 1500
- beSync1 on VWS1 and beSync1 on VWS2 use port 1501

Sync files

Files are named with the form “sync-VWARS-YYYYMMDDHHMMSS-UUU” where:

- UUU is a unique number used to differentiate when two or more files are produced in a second
- VWARS is the beVWARS number to distinguish between beVWARS instances.
The files are stored in numbered directories within the main sync file directory. The name of the subdirectories follows the beVWARS number which wrote the sync file. This is to aid the beSync during recovery of individual beVWARS instances.

**Synchronization diagram**

This diagram shows the processes and data involved in the synchronization.

**Synchronization process**

This table describes how the VWS keeps the Voucher and Wallet Servers in a pair in sync during an update or reservation.

**Note:** This process starts at the point the beVWARS has triggered all the configured event plug-ins and message handlers and is ready to start a reservation or update.

<table>
<thead>
<tr>
<th>Stage</th>
<th>Description</th>
</tr>
</thead>
</table>
| 1     | For each reservation or update, beVWARS:  
  - Updates the cache  
  - Writes the update or reservation to the sync file  
  - Updates the sync file sequence number  
  If there is no sync file, the beVWARS creates one. |
| 2     | If the request is an update, the beVWARS on the primary VWS writes the EDR to record the transaction. |
| 3     | beVWARS closes its sync file after a configurable period or a configurable number of entries, or force-closes it (if a commit message is received). It writes the current transaction set to the file and performs a file flush on the current file to ensure the last transaction set is written safely to disk. |
Chapter 1

### Resynchronizations

This table describes the stages involved in resynchronizations.

<table>
<thead>
<tr>
<th>Stage</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>If a resynchronization is triggered, the beSync on the recovering VWS queries a local beVWARS for its last update numbers (both local and remote update numbers).</td>
</tr>
<tr>
<td>2</td>
<td>beSync on the recovering compares the sequence number from the local beVWARS with the last local update sequence number to check whether any local updates have been missed. If the numbers do not match, beSync streams all local uncommitted transactions to the recovering beVWARS.</td>
</tr>
<tr>
<td>3</td>
<td>When all local updates have been sent, beSync sends a &quot;request all reservations&quot; message to the beSync on the running VWS specifying which beVWARS instance to update.</td>
</tr>
</tbody>
</table>
Chapter 1

Wallet and Bucket Expiry

Introduction

Like most functions, wallet expiry and bucket expiry and removal are triggered when a wallet is loaded. VWS uses beVWARSExpiry to control when wallet expiry events are triggered. Additional wallet expiry processing can be done by plug-ins and processes provided by other applications. VWS provides basic bucket handling, though this functionality can be extended by plug-ins which are triggered on bucket expiry or bucket delete/removal events.

Note: Expiry handling is optional. If no expiry dates are configured for wallets and/or buckets, no expiry handling will be executed.

Wallet management processes

This table describes the main components in VWS.

<table>
<thead>
<tr>
<th>Process</th>
<th>Role</th>
<th>Further information</th>
</tr>
</thead>
<tbody>
<tr>
<td>beServer</td>
<td>Handles connections from client processes (including BeClient processes) and controls routing to beVWARS processes.</td>
<td>beServer (on page 58)</td>
</tr>
<tr>
<td>beGroveller</td>
<td>beGroveller triggers processing on wallets which have not been triggered by a subscriber action. This enables VWS to ensure required actions are taken against all wallets and buckets.</td>
<td>beGroveller (on page 52)</td>
</tr>
</tbody>
</table>
| beVWARS   | beVWARS is the core of the VWS. More than one beVWARS will usually be running on a VWS. It does the following:  
  - Reads and caches wallet and voucher information from the E2BE database  
  - Manages all queries, reservations and updates against wallets | beVWARS (on page 85) |
<table>
<thead>
<tr>
<th><strong>Chapter 1</strong></th>
<th><strong>NCC Voucher and Wallet Server Technical Guide</strong></th>
</tr>
</thead>
</table>
| - Manages all queries, redemptions and state changes for vouchers  
  - Writes sync files  
  - Writes EDRs | |
| **beVWARSExpiry** | beVWARSExpiry monitors wallets and buckets, checking for wallets and buckets which have passed their expiry date. If it finds a wallet or bucket which requires expiring, it processes the record as configured and triggers any Expiry plug-ins with a Wallet Event or Bucket Event. **beVWARSExpiry** (on page 105)  
**beVWARS plug-ins** (see "beVWARS plugins" on page 7) |
| **Expiry plug-ins** | beVWARSExpiry starts an expiry event when it finds an expired wallet or bucket. Each expiry event can trigger one or more expiry plug-ins. Each expiry plug-in will take its own action. **Technical guide for the application using VWS. For an example, see CCS Technical Guide.** |
| **E2BE database** | The databases on the VWSs. They hold a subset of the data from the SMF. |

**Expiry diagram**

This diagram shows the basic processes, communication and relationships for expiring wallets and buckets.
Wallet and bucket expiry processing

This table describes how wallets and buckets are expired when they have passed their expiry date.

<table>
<thead>
<tr>
<th>Stage</th>
<th>Description</th>
</tr>
</thead>
</table>
| **1** | *beVWARS* (on page 85) loads a wallet. Loading a wallet can be triggered by one of the following:  
- A query, request or reservation from a requesting process as part of normal processing  
- *beVWARS* processing a wallet from a list of wallet IDs to grovel from *beGroveller* (on page 52). |
| **2** | Loading the wallet triggers *beVWARSExpiry* (on page 105). |
| **3** | For wallets which are Active or Dormant, *beVWARSExpiry* checks whether the wallet has passed its expiry date.  
*Tip:* Wallet expiry date checking is defined by *expireAtMidnightTZ* (on page 105).  
If *expireAtMidnightTZ* is set, *beVWARSExpiry* will expire the wallet and buckets the next time they are loaded after the midnight in the specified time zone which follows the expiry date.  
If the wallet has not expired, *beVWARSExpiry* checks whether any of the buckets in the wallet have passed their expiry date. For each bucket which has passed its expiry date, *beVWARSExpiry* deletes the bucket. If two buckets expire at exactly the same time, the buckets will be processed in bucket ID order.  
This triggers any *beVWARS* event plug-ins which are designed to handle Bucket Expiry events.  
*Note:* If a bucket is expired, it will be deleted unless an event plug-in provides specific logic which retains the bucket.  
*Example:* *ccsVWARSPeriodicCharge* processes expiring periodic charge buckets. It keeps the periodic charge bucket and sets the expiry date to a point in the future. |
| **4** | If the wallet has passed its expiry date, *beVWARSExpiry* sets the wallet’s state to terminated and fires a Wallet Expiry event. This triggers any *beVWARS* event plug-ins which are designed to handle Wallet Expiry events.  
Expiry plug-ins which handle Wallet Expiry events can be provided as part of another application such as *CCS*.  
*Example:* When triggered by a Wallet Expiry event, *ccsVWARSExpiry* writes an EDR and adds the wallet to a list of expired wallets which is used to update the HLR records. |
| **5** | *beVWARSExpiry* checks the configuration.  
If *expireBucketsForExpiredWallets* (on page 106) is set to true, it deletes all buckets with a positive or zero value. If *expireNegativeBuckets* (on page 106) is also set to true, *beVWARSExpiry* will also delete buckets with negative values.  
This triggers any *beVWARS* event plug-ins which are designed to handle Bucket Expiry events. |
| **6** | If *removeEmptyBuckets* (on page 107) is set to true, *beVWARSExpiry* deletes all buckets with a 0 balance from the E2BE.  
*Exception:* If the last bucket in a wallet which has not expired has a value of 0, that bucket will be left.  
For more information about the expiry plug-ins provided with other applications, see the application’s technical guide. |
BE States

Introduction

In a VWS domain, there are two Voucher and Wallet Servers, in a redundant configuration. If one VWS in a pair is down, the system will work as normal. When the other peer comes back up:

1. It will resynchronize with the uninterrupted peer without prompting
2. Service will continue as if nothing happened.

If the network link between BeClients and BeServers (or between VWSs in a domain) is disconnected, the BeClients that can see at least one member of a VWS pair should be able to keep running. When connectivity is restored, changes made to records held on the VWSs are resynchronized.

BE states

The beVWARS is responsible for maintaining the current state of a Voucher and Wallet Server. There are three possible states for a VWS:

- Running
- Recovering
- Disabled

Running

This is the normal state of a VWS.

In this state:

- beSync is streaming updates and receiving streamed updates
- beServer is accepting connections from BeClients and processing requests

beSync listens on a defined TCP port, for connections from the beSync on the peer VWS. When this connection is open and streaming, all reservations and transactions are sent to the other VWS.

Disabled

This is the initial state of a VWS, and it may return to this state in a variety of failure scenarios.

In this state:

- beServer does not accept any connections from BeClients
- beSync does not accept any connections from the peer VWS
- The beGroveler does not run
- No internal processing is performed

beVWARS failure

When the SLEE watchdog notices a beVWARS process has failed, the beServer will:

- Cease to read new work from the client sockets
- Allow the remaining active beVWARS instances to quiesce
- Close all client connections (when all beVWARS instances are idle)

When all connections are closed, the BeClient processes will failover to the secondary Voucher and Wallet Server. The local beServer removes all existing context and beVWARS routes for the failed beVWARS. These are recovered during the beVWARS recovery, which delivers all context from the remote VWS.
Chapter 1

Recovering

In this state:

- The beGroveller does not run
- Synchronization can be in any state
- beServer should not be accepting new connections from the BeClients.

beServer begins in a recovery state expecting to receive all of its contexts from the beServer on the peer, and getting the OK from all local beVWARS indicating they are in sync before accepting client connects and client requests.

On VWS recovery/startup, the local sync files are processed to ensure there are no lost local updates. Then a connection is made to the peer VWS, to request all updates since the last remote update received. For more information about this process, see Resynchronizations (on page 16).

If one VWS is disabled for an extended period of time, its peer will amass a significant number of updates in the sync directory specified by spoolDirectory (on page 71) (typically in the /IN/service_packages/E2BE/sync/ directory). When the VWS is re-enabled, the updates will be requested and the VWSs will return to a synchronized state.

beVWARS recovery

Individual beVWARS processes recover independently. If a beVWARS process fails, the other beVWARS processes do not detect this, and continue to function (though, due to the beServer disabling connections, they will only be processing remote transactions from the now active secondary VWS).

The failed beVWARS independently goes through its recovery process along with the beSync process, until it is able to move back into running state. The beServer asks the remote beServer for all contexts for the beVWARS which failed.

State transitions

Here is what a user may expect to see, in the transition of a Voucher and Wallet Server from one state to another.

Note: The initial state of the VWS should be disabled.

<table>
<thead>
<tr>
<th>Transition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Disabled to Recovering</td>
<td>beServer should prepare to accept contexts.</td>
</tr>
<tr>
<td></td>
<td>beVWARS should prepare to send contexts to the beServer, and should prepare to receive Operations. This should include a complete new Reservation load, so all existing reservations should be erased.</td>
</tr>
<tr>
<td></td>
<td>beVWARS will not request work from beGroveller.</td>
</tr>
<tr>
<td></td>
<td>beGroveller will not return wallet IDs for groveling to beVWARS.</td>
</tr>
<tr>
<td></td>
<td>beSync should initiate recovery.</td>
</tr>
<tr>
<td>Disabled to Running</td>
<td>Not a possible transition.</td>
</tr>
<tr>
<td>Recovering to Running</td>
<td>beServer should start accepting connections.</td>
</tr>
<tr>
<td></td>
<td>beVWARS can start to accept new requests and can start to send requests to beGroveller for lists of wallet IDs to grovel.</td>
</tr>
<tr>
<td></td>
<td>beGroveller will determine if it should run. If it should, it will start to return wallet IDs to grovel to beVWARS. For more information about the beGroveller, see beGroveller (on page 52).</td>
</tr>
<tr>
<td></td>
<td>beSync should proceed as it was (it usually leads the recovery.</td>
</tr>
<tr>
<td>Transition</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| Recovering to Disabled | beServer should terminate all connections.  
beVWARS should disable the beGroveller.  
beVWARS should stop requesting work from beGroveller.  
beGroveller will stop accepting requests from beVWARS for wallet IDs to grovel. an inactive state. |
| Running to Recovering   | beServer should terminate all connections.  
beVWARS should stop requesting work from beGroveller.  
beGroveller will stop accepting requests from beVWARS for wallet IDs to grovel.  
beSync should disconnect open connections and initiate recovery. |
| Running to Disabled     | beServer should terminate all connections.  
beVWARS should stop requesting work from beGroveller.  
beGroveller will stop accepting requests from beVWARS for wallet IDs to grovel.  
beSync should close all open connections and return to an inactive state. |

### EDR Processing

#### Introduction

Each Voucher and Wallet Server in a domain logs EDRs for all actions which are successfully completed on the local VWS.

#### EDR processing diagram

This diagram shows how EDRs are processed by VWS.
**VWS EDR processing**

This process describes how VWS processes EDRs.

<table>
<thead>
<tr>
<th>Stage</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>beVWARS receives an update request from the local beServer.</td>
</tr>
<tr>
<td>2</td>
<td>beVWARS updates the relevant cache and queues the EDR write until the next flush. For more information about queuing and flushing, see <em>Queuing and flushing updates</em> (on page 12).</td>
</tr>
</tbody>
</table>
| 3     | When the next flush is triggered, beVWARS:  
  - Writes all queued EDR records to a new EDR file  
  - Records the EDR file name in the E2BE database (in the BE_CDR_FILE table).  
  
  **Note:** Entering the EDR file name in the BE_CDR_FILE table indicates that the EDRs in the file should be:  
  - Accepted by the rest of the system  
  - Transmitted to the SMS for consolidation into the SMF database. |
| 4     | beCDRMover moves completed EDR files from the working directory to the output directory. |
| 5     | cmnPUSHFiles transfers the EDR file to the SMS. |
EDR triggers

EDRs are written on the Voucher and Wallet Servers when a wallet or voucher is modified. The following messages, among others, cause the beVWARS to write EDRs:

- Call End Notification
- Wallet Recharge Request
- Named Event

MFile Updates

Introduction

The MFile contains a subset of the Voucher and Wallet Server data, used to reduce network traffic on the system. Some of the information held within the VWS changes less frequently, such as Tariffs. It is this data which is copied to the MFile and held on the VWS. The system reads this MFile, enabling it to retrieve data quickly, thereby reducing network traffic to the Voucher and Wallet Servers.

MFile data types

A MFile will need to be recompiled if any of the following data types are changed:

- Discount Period
- Discount Sets
- Geography Sets
- Billing Periods
- CLIXDN Mappings
- Tariff Plans
- Product Types
- Tariff Plans
- Currency
Update process diagram

Here is an example showing an update to an MFile.

Image of the update process diagram.

Update process - mfile

This table describes the process through which MFiles are updated.

<table>
<thead>
<tr>
<th>Stage</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 - 4</td>
<td>Through the Prepaid Charging user interface (UI) on the SMS node, the system administrator updates the details contained in the MFile and clicks <strong>Save</strong>.</td>
</tr>
<tr>
<td>5</td>
<td>A new entry is added to the CCS_MFILE table in the SMF database.</td>
</tr>
</tbody>
</table>
The relevant tables in the SMF are updated and the data is transferred to the VWS nodes using replication.

When the new CCS_MFILE entry arrives on the BE, VWS sends a notification to the ccsMFileCompiler.

The ccsMFileCompiler updates the MFile file name table (CCS_MFILE) in the BE database.

ccsMFileCompiler then generates a new MFile from the updated data in the E2BE database.

If ccsMFileCompiler has not already connected to the beServer, it uses the beLocationPlugin (on page 35) to extract the location of the beServer from the BE database. After establishing the connection, or if it is already connected, ccsMFileCompiler sends a request to the beServer to reload the MFile.

The ccsMFileLoadHandler message handler on the beServer forwards the reload request to the ccsMFileLoader message handler in beVWARS.

beVWARS uses ccsMFileLoader to reload the new MFile.

For more information about the ccsMFile processes, see *CCS Technical Guide*.

## Statistics

### Introduction

VWS statistics are generated by each VWS VWS, and then transferred at periodic intervals to the SMS for permanent storage and analysis.

VWS also records statistics for applications which use the VWS, such as CCS. For more information about these statistics, see the application's technical guide.

### SMS statistics subsystem

The statistics system provided by SMS provides the functionality which collects the statistical events logged by VWS processes.

For more information about the SMS statistics subsystem, see *SMS Technical Guide*.

### Collected statistics

This table describes the statistics produced by VWS processes.

<table>
<thead>
<tr>
<th>Statistic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NUM_TOTAL_REQ</td>
<td>Total number of requests sent to the VWS.</td>
</tr>
</tbody>
</table>

*Note: All statistics are collected with a period of 1800 seconds.*

For more information about the request messages these statistics measure, see *Supported requests* (on page 9).
Chapter 2
Configuration

Overview

Introduction

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

In this chapter

This chapter contains the following topics.

Configuration Overview 27
Configuring the Environment 28
eserv.config Configuration 28
BE Shared Parameters 34
User Interface-Based Configuration Tasks 38
SLEE.cfg 39

Configuration Overview

Introduction

This topic provides a high level overview of how the VWS application is configured.

Note: There are several configuration options that are not explained in this chapter. These options should not be changed by the user without first consulting Oracle for technical support.

Configuration process overview

This table describes the steps involved in configuring a VWS for the first time.

<table>
<thead>
<tr>
<th>Stage</th>
<th>Description</th>
</tr>
</thead>
</table>
| 1     | The environment that VWS will run in must be configured correctly. This includes:  
        • If the directory VWS was installed into was not the recommended directory (/IN/service_packages/E2BE), setting the root directory  
        • Configuring the location of the EDR directories |
| 2     | The eserv.config file must be configured for the following machines:  
        • SMS nodes  
        • SLC nodes  
        • VWS nodes  
        The example file should be copied into the main eserv.config, and any mandatory parameters configured. The parameters which must be set are listed at the top of the eserv.config. For more information, see eserv.config Configuration (on page 28). |
| 3     | The screen-based configuration tasks must be completed through the CCS User Interface (UI). |
| 4     | The SLEE.cfg file must contain references to the VWS SLEE applications and interfaces. |
Configuration components

VWS is configured by the following components:

<table>
<thead>
<tr>
<th>Component</th>
<th>Locations</th>
<th>Description</th>
<th>Further Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>eserv.config</td>
<td>all SMSs and VWSs</td>
<td>VWS is configured by the BE section of eserv.config.</td>
<td>eserv.config Configuration</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(on page 28)</td>
</tr>
<tr>
<td>SLEE.cfg</td>
<td>all VWSs</td>
<td>SLEE.cfg sets up SLEE interfaces and applications.</td>
<td>SLEE.cfg (on page 39)</td>
</tr>
<tr>
<td>Domains screen</td>
<td>SMF database</td>
<td>Domains must be set up which define the Voucher and Wallet Servers, and available services in the Domain screen in the CCS UI.</td>
<td>Screen-Based Configuration Tasks (see &quot;User Interface-Based Configuration Tasks&quot; on page 38)</td>
</tr>
</tbody>
</table>

Configuring the Environment

Oracle environment variables

The VWS UNIX system account ebe_oper requires the standard ORACLE environment variables to be present.

eserv.config Configuration

Introduction

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

It is located in the following directory:

/IN/service_packages/

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

Configuration file format

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

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

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

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

or

```{ name="route6"```
Chapter 2

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

or

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

**eserv.config files delivered**

Most applications come with an example `eserv.config` configuration in a file called `eserv.config.example` in the root of the application directory, for example, `/IN/service_packages/eserv.config.example`.

**Editing the file**

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

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

**Loading eserv.config changes**

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

**Example eserv.config configuration**

This is an example of the `BE` section of an `eserv.config` file (comments have been removed). It is not intended to be used in a production environment, but only to illustrate the configurations available.

Details on the configurations in this file are located in various sections later in this guide.

```javascript
BE = {
    serverId = 11
    amPrimary = true
    oracleUserAndPassword="/"
    beLocationPlugin = "libGetccsBeLocation.so"
    soloBE = true
    timerIfName = "Timer"
    enableGroveling = true
```
freeDiskSpaceWarningThres = 100
freeDiskSpaceShutdownThres = 10
freeDiskSpaceCheckInterval = 300
lowDiskSpaceNotificationInterval = 30

beServer = {
clientSelectTime = 1000000
quiesceLength = 100000
serverPortOverride = 1500
clientSocketBufferSize = 10240
maxDownstreamQueueLength = 1000
downstreamOverloadSleepUSec = 100000
dbConnCheckTime = 5
recoveryReportInterval = 60
shutdownDelayTime = 4
startupRetryPeriodSeconds = 2

notEndActions = [
  {type="IR ", action="ACK "}
  {type="SR ", action="ACK "}
  {type="SR ", action="NACK"}
  {type="INER", action="ACK "}
  {type="SNER", action="ACK "}
  {type="SNER", action="NACK"}
]

handlers = [
  "libbeServerPingPlugin.so"
]

messageRoutingPlugins = [
  "libbeMsgRouterDefault.so"
]

msgRouterDefault = {
  roundRobinTypes = [
    "VI 
  ]
  routeOnVoucherNumber = true
}

purge = {
  purgeInterval = 300
  vwarsTimeout = 10
  expectedKeep = 60
  noExpectedKeep = 3600
}

routingVoucherNumberLength = 10

beVWARS = {
  voucherReservationPeriodSeconds = 120
  useTimeFromClient = true
  maxTransactionsPerSet = 7
  maxOpenDialogTime = 5.0
  maxDownstreamQueueLength = 10000
  downstreamOverloadSleepUSec = 100000
  minResyncReservationLength = 5
  createBucketExpiryDays = 30

  walletCache = {

maxSize = 100000
maxLoopSize = 500
checkBeforeFlush = false
}

voucherCache = {
    maxSize = -100000
    maxLoopSize = 500
    checkBeforeFlush = false
}
groveller = {
    periodMsec = 1200
    requestHighWaterMark = 1
    walletLowWaterMark = 100
    requestTimeout = 300
    peerDatabaseLogin = ""
    peerWalletCheckRetrySeconds = 60
}
duplicateDetection = {
    keepDirectSeconds = 60.0
    keepSyncSeconds = 60.0
    directMaxDelaySeconds = 1.0
    syncMaxDelaySeconds = 1.0
}

setLastActivationDateStates = [
    [PREU]
]

plugins = [
    "beVWARSEpiry.so"
]

handlers = [
    "beVWARSCCDRHandler.so"
]

syncWriter = {
    maxRecordsPerFile = 100
    maxSecondsPerFile = 2
}

dbWriter = {
    flushPeriod = 10
    cdrOutputDirectory = "/IN/service_packages/E2BE/logs/CDR"
    balanceCreateBufferSize = 1000
    balanceUpdateBufferSize = 1000
    balanceDeleteBufferSize = 1000
    bucketCreateBufferSize = 1000
    bucketUpdateBufferSize = 1000
    bucketDeleteBufferSize = 1000
    walletCreateBufferSize = 1000
    walletUpdateBufferSize = 1000
    walletDeleteBufferSize = 1000
    voucherCreateBufferSize = 1000
    voucherUpdateBufferSize = 1000
    voucherDeleteBufferSize = 1000
}

tracing = {""}
enabled = true
dbugLevel = "all"
walletIds = [  
    <walletid1>,  
    <walletid2>
]
beClients = [  
    "<Beclient1>",  
    "<Beclient2>"  
]
)
}
) # BE.beVWARS
beVWARSExpiry = {  
    expireNegativeBuckets = false  
    removeEmptyBuckets = false  
    expireBucketsForExpiredWallets = false  
    expireAtMidnightTZ = "Asia/Vladivostok"
}
beVWARSMergeBuckets = {  
    maxBuckets = -1  
    triggerPlugins = false
}
beCDRMover = {  
    oracleService = ""
    oracleUser = "e2be_admin"
    oraclePassword = "password"
    outDirectory = "/IN/service_packages/E2BE/logs/CDR"
    destinationDirectory = "/IN/service_packages/E2BE/logs/CDR-out"
    timeout = 600  
    numberOfRecordsToCommit = 10
}
)
)
# BE.beCDRMover
cmnPushFiles = {  
    CDR = [  
        "-d", "/IN/service_packages/E2BE/logs/CDR-out"
        "-r", "/IN/service_packages/CCS/logs/CDR-in"
        "-h", "smp1hostname"
        "-F"
    ]
}
beSync = {  
    shared = {  
        noWorkSleepTime = 0.2  
        spoolDirectory = "/IN/service_packages/E2BE/sync"
        spoolChunkSize = 16
        badFileDirectory = "/IN/service_packages/E2BE/tmp"
        maxDownstreamQueueLength = 10000  
        downstreamOverloadSleepUSec = 100000
    }
    sink = {  
        inSyncThresholdSeconds = 5  
        inSyncReportingPeriodRecords = 10000  
        retryConnectionDelaySeconds = 30  
        maxRetriesBeforeStart = 5  
        localUpdateChunkSize = 100
        heartbeatPeriodSeconds = 10
    }
}
source = {
    recordSendingChunkSize = 50
    maxQueueLength = 50
}

BeClient = {
    clientName = "scpClient"
    heartbeatPeriod = 10000000
    maxOutstandingMessages = 100
    connectionRetryTime = 2

    plugins = [
        {
            config="",
            library="libclientBcast.so",
            function="makeBroadcastPlugin"
        }
    ]

    notEndActions = [
        {type="IR ", action="ACK "}
        {type="SR ", action="ACK "}
        {type="SR ", action="NACK"}
        {type="INER", action="ACK "}
        {type="SNER", action="ACK "}
        {type="SNER", action="NACK"}
    ]

    billingEngines = [
        {
            id = 1,
            primary  = { ip="123.123.123.123", port=1500 },
            secondary = { ip="123.123.123.124", port=1500 }
        }
    ]

    beGroveller = {
        quorumHost = "produsms-cluster"
        maxIDsPerResponse = 160
        retrySeconds = 60
        processExpiredBuckets = true
        noProcessingTimes = [
            { startsAt = "06:00", endsAt = "09:30" }
            { startsAt = "11:30", endsAt = "14:00" }
            { startsAt = "16:00", endsAt = "21:00" }
        ]
        connectionRetryTime = 60
        heartbeatPeriod = 300000000
        filledBufferThreshold = 480
        ludProcessingTime = "14:04"
    }

    triggering = {
        Control_Plan = "Reward"
        Service_Handle = "CCS_BPL"
        CCSNamespace = "http://eng-prf-zone01-zl/wsdls/ON/CCSNotifications.wsdl"
        Username = "ube"
        Password = "password"
        edr = false
    }
BE Shared Parameters

Purpose

The BE section of the eserv.config file for the VWS contains parameters that are shared by various VWS background processes. These parameters define the settings that are common to the background processes.

Configuration

VWS accepts these parameters from eserv.config.

```plaintext
serverId = int
amPrimary = true|false
oracleUserAndPassword="/"
beLocationPlugin = "lib"

soloBE = true|false

timerIfName = "str"

enableGrovelling = true|false

freeDiskSpaceWarningThres = MB
freeDiskSpaceShutdownThres = MB
freeDiskSpaceCheckInterval = secs
lowDiskSpaceNotificationInterval = secs
```

Example BE shared parameters configuration

The following section sets the shared BE configuration parameters.

```plaintext
BE = {
    serverId = 11
    amPrimary = true
    oracleUserAndPassword="/"
    beLocationPlugin = "libGetccsBeLocation.so"

    soloBE = true

    timerIfName = "Timer"

    enableGrovelling = true

    freeDiskSpaceWarningThres = 100
    freeDiskSpaceShutdownThres = 10
}
```
Parameters

Here are the available shared VWS parameters.

**amPrimary**

**Syntax:** `amPrimary = true|false`

**Description:** True if this is the primary VWS in the pair.

**Type:** Boolean

**Optionality:** Optional, default used if not set

**Allowed:**

**Default:** true

**Notes:**

**Example:** `amPrimary = false`

**beLocationPlugin**

**Syntax:** `beLocationPlugin = "lib"`

**Description:** The plug-in library that finds the Voucher and Wallet Server details of the Voucher and Wallet Servers to connect to.

**Type:** String

**Optionality:** Optional (default used if not set)

**Allowed:**

**Default:** `libGetccsBeLocation.so`

**Notes:** This library must be in the LD_LIBRARY_PATH.

**Example:** `beLocationPlugin = "libGetccsBeLocation.so"`

**enableGrovelling**

**Syntax:** `enableGrovelling = true|false`

**Description:** Whether or not to process wallets when spare resources are available.

**Type:** Boolean

**Optionality:** Optional (default used if not set)

**Allowed:**

**true**

Use beGroveller to trigger events on wallets and balances which are not used often.

**false**

Do not do background triggering of events on wallets and balances.

**Default:** true

**Notes:**

**Example:** `enableGrovelling = true`

**oracleUserAndPassword**

**Syntax:** `oracleUserAndPassword = "usr/pwd"`

**Description:** The Oracle user and password for the connections to the E2BE database for VWS processes.

**Type:** String
### serverId

**Syntax:**

```
serverId = id
```

**Description:** The ID of the VWS pair.

**Type:** Integer

**Optionality:** Optional, default used if not set

**Allowed:**

- Default: 1

**Notes:** Set to 1 if this is not a VWS

**Example:**

```
serverId = 11
```

### soloBE

**Syntax:**

```
soloBE = true|false
```

**Description:** Set to true if there is no peer VWS node to synchronize with in this billing engine domain.

**Type:** Boolean

**Optionality:** Optional (default used if not set)

**Allowed:** true, false

**Default:** false

**Notes:**

**Example:**

```
soloBE = true
```

### timerIfName

**Syntax:**

```
timerIfName = "name"
```

**Description:** The name for the SLEE Timer interface component.

**Type:** String

**Optionality:** Optional, default used if not set

**Allowed:**

- Default: Timer

**Notes:** Must match the handle of the timer interface in SLEE.cfg. For more information about the SLEE, see SLEE Technical Guide.

**Example:**

```
timerIfName = "Timer"
```

### freeDiskSpaceCheckInterval

**Syntax:**

```
freeDiskSpaceCheckInterval = seconds
```

**Description:** How often (in seconds), beServer will check whether there is more than freeDiskSpaceWarningThres (on page 37) space free on the disk. If there is less than freeDiskSpaceShutdownThres (on page 37), beServer will close its connections and stop accepting requests.

**Type:** Integer

**Optionality:** Optional (default used if not set)
Allowed: 0  Disables the disk space check
        positive integer  Checks the available disk space at the specified interval

Default: 300
Notes: If a shutdown is triggered, beServer will log a critical-level error to the syslog.
Example: freeDiskSpaceCheckInterval = 300

freeDiskSpaceShutdownThres
Syntax:  freeDiskSpaceShutdownThres = MB
Description: When free disk space in MB is below this threshold beServer closes its connections and stops taking new requests. Threshold applies to the partitions containing the directories set by the following parameters:
            •  spoolDirectory (on page 71)
            •  cdrOutputDirectory (on page 99)
Type: Integer
Optionality: Optional (default used if not set)
Allowed: 0  Disables the threshold shutdown
        positive integer  The MB threshold
Default: 10
Notes: When the disk space has come back above the threshold, the beServer will re-enable, and go back to running state. If beServer stops taking requests, it will log a Critical-level error to the syslog. Disk space is checked by beServer at the frequency set by freeDiskSpaceCheckInterval (on page 36).
Example: freeDiskSpaceShutdownThres = 10

freeDiskSpaceWarningThres
Syntax:  freeDiskSpaceWarningThres = MB
Description: Low disk space threshold, in Megabytes, for the partitions containing the directories set by the following parameters:
            •  spoolDirectory (on page 71)
            •  cdrOutputDirectory (on page 99)
Type: Integer
Optionality: Optional (default used if not set)
Allowed: 0  Disables the threshold warning
        positive integer  The MB threshold
Default: 100
Notes: If the threshold is reached, an Error-level warning is logged to the syslog. Disk space is checked by beServer at the frequency set by freeDiskSpaceCheckInterval (on page 36).
Example: freeDiskSpaceWarningThres = 200
lowDiskSpaceNotificationInterval

Syntax:  

lowDiskSpaceNotificationInterval = seconds

Description:  
The number of seconds between logging the error triggered by the low disk space 
warning or shutdown thresholds being triggered.

Type:  

Integer

Optionality:  

Optional (default used if not set)

Allowed:  

Default:  

30

Notes:  
The threshold is set by freeDiskSpaceWarningThres (on page 37) and 
freeDiskSpaceShutdownThres (on page 37).  
The notification interval should be set to a value higher than 
freeDiskSpaceCheckInterval (on page 36), as it will only log an error if the 
check interval has recorded a low disk space condition since the last error was 
logged.

Example:  

lowDiskSpaceNotificationInterval = 30

Deprecated SLEE Name Definitions

The parameters listed in this section have been deprecated and should not be used. You should delete 
them from the BE section of eserv.config if they are currently defined.

grovellerIfNamePrefix

Syntax:  
grovellerIfNamePrefix = "name"

Default:  
beGroveller

Example:  
grovellerIfNamePrefix = "beGroveller"

serverIfName

Syntax:  
serverIfName = "name"

Default:  
beServer

Example:  
serverIfName = "beServer"

syncIfName

Syntax:  
syncIfName = "name"

Default:  
beSync

Example:  
syncIfName = "beSync"

vwarsIfNamePrefix

Syntax:  
vwarsIfNamePrefix = "name"

Default:  
beVWARS

Example:  
vwarsIfNamePrefix = "beVWARS"

User Interface-Based Configuration Tasks

Introduction

These procedures are normally performed only once, after the installation and initial configuration of the 
system.
For more information about accessing the CCS screens, see CCS User's Guide.

**Defining VWS locations**

The system requires the location of VWS machines to be defined. These are defined using the New Domain or Edit Domain screens, accessed from the Service Management screen.

For more information about configuring domains, see CCS User's Guide.

### SLEE.cfg

**About Configuring VWS SLEE Interfaces**

The VWS includes the beVWARS, beSync, and beServer SLEE interfaces that run on the VWS nodes. For these processes to run correctly, they must be configured in the SLEE.cfg file. The SLEE is automatically configured during installation to run one or more instances of each by the following lines in SLEE.cfg:

```
INTERFACE=beVWARS  beVWARSStartup.sh /IN/service_packages/E2BE/bin instance_count EVENT
INTERFACE=beSync   beSyncStartup.sh  /IN/service_packages/E2BE/bin instance_count EVENT
INTERFACE=beServer beServerStartup.sh /IN/service_packages/E2BE/bin instance_count EVENT
```

Where `instance_count` is the number of instances of the interface process to run.

**Note:** The actual startup script names may vary.

You should only update this configuration if you want to change the number of instances to run of any of these processes. For example, if there is a high volume of traffic on the VWS, you may be able to improve performance by running additional instances of these processes.

For more information about configuring SLEE interfaces, see the discussion on Configuring the SLEE in SLEE Technical Guide.

**About Configuring MAXEVENTS**

The value of MAXEVENTS sets the maximum number of event objects that the system can hold in shared memory. If MAXEVENTS is exceeded when the system is running, no more events or calls will be accepted and alarm messages will be sent. This means that you should set MAXEVENTS to a value that is big enough to handle an overload situation. You can estimate this value by using the following formula:

\[
\text{MAXEVENTS} = (\text{num}\_\text{beServers} \times \text{max}\_\text{beServer}\_\text{queue}) + (\text{num}\_\text{beVWARS} \times \text{max}\_\text{beVWARS}\_\text{queue}) + \\
(\text{num}\_\text{beSyncs} \times \text{max}\_\text{beSync}\_\text{queue}) + \text{contingency}
\]

Where:

- `num\_beServers` is the number of instances of the beServer interface defined in the SLEE.cfg configuration file.
- `max\_beServer\_queue` is the maximum number of beVWARS response events that may be queued up for the beServer. This is the value specified for the BE.beVWARS.maxDownstreamQueueLength parameter in the eserv.config configuration file.
- `num\_beVWARS` is the number of instances of the beVWARS interface defined in the SLEE.cfg configuration file.
- `max\_beVWARS\_queue` is the value specified in the eserv.config configuration file for either BE.beSync.maxDownstreamQueueLength or BE.beServer.maxDownstreamQueueLength, whichever value is greater.
- `num\_beSyncs` is the number of instances of the beSync interface defined in the SLEE.cfg configuration file.
max_beSync_queue is the maximum number of beVWARS events that may be queued up for the beSync. This is the value specified for the `BE.beVWARS.maxDownstreamQueueLength` parameter in the `eserv.config` configuration file.

contingency is an estimated value for any additional VWS events such as VWS control messages. A typical value for contingency would be 5000.

Example

This example shows how to calculate the value for MAXEVENTS for one beServer interface, six instances of the beVWARS interface, and two instances of the beSync interface, and where:

- `BE.beVWARS.maxDownStreamQueueLength = 10000`
- `BE.beSync.maxDownStreamQueueLength = 50000`
- `BE.beServer.maxDownStreamQueueLength = 50000`
- `contingency = 5000`

MAXEVENTS = (1 * 10000) + (6 * 5000) + (2 * 10000) + 5000 = 65000

You configure MAXEVENTS in the `SLEE.cfg` configuration file. For more information about configuring MAXEVENTS, see the discussion on Configuring the SLEE in *SLEE Technical Guide*.

Loading SLEE.cfg changes

If you change the `SLEE.cfg` file, then you must restart the SLEE to enable the new options to take effect.

For more information about restarting the SLEE, see *SLEE Technical Guide.*
Chapter 3

Background Processes

Overview

Introduction

This chapter explains the processes which run automatically as part of the application. These processes are started automatically by one of the following:

- inittab
- crontab
- Service Logic Execution Environment SLEE

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

In this chapter

This chapter contains the following topics.

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>beCDRMover</td>
<td>41</td>
</tr>
<tr>
<td>BeClient</td>
<td>44</td>
</tr>
<tr>
<td>beGroveller</td>
<td>52</td>
</tr>
<tr>
<td>beServer</td>
<td>58</td>
</tr>
<tr>
<td>beSync</td>
<td>69</td>
</tr>
<tr>
<td>beServiceTrigger</td>
<td>75</td>
</tr>
<tr>
<td>beVWARS</td>
<td>85</td>
</tr>
<tr>
<td>beVWARSFCDRHandler</td>
<td>104</td>
</tr>
<tr>
<td>beVWARSExpiry</td>
<td>105</td>
</tr>
<tr>
<td>beVWARSMergeBuckets</td>
<td>107</td>
</tr>
<tr>
<td>cmnPushFiles</td>
<td>109</td>
</tr>
<tr>
<td>Event Storage Interface</td>
<td>112</td>
</tr>
<tr>
<td>libbeMsgRouterDefault</td>
<td>114</td>
</tr>
<tr>
<td>libBeClientIF</td>
<td>114</td>
</tr>
<tr>
<td>libclientBcast</td>
<td>115</td>
</tr>
<tr>
<td>libbeEventFactory</td>
<td>115</td>
</tr>
</tbody>
</table>

beCDRMover

Purpose

beCDRMover moves completed EDR files from the working directory to a directory from which they are copied to the SMS. The inter-machine transfer is completed by cmnPushFiles.

Startup

This task is started by entry be_1 in the inittab, via the shell script:

```
/IN/service_packages/E2BE/bin/beCDRMoverStartup.sh
```

Note: The above is a default and may vary as per configuration.
Configuration

beCDRMover accepts the following parameters from eserv.config.

```json
beCDRMover = {
    oracleService = "str"
    oracleUser = "name"
    oraclePassword = "str"
    outDirectory = "dir"
    destinationDirectory = "dir"
    timeout = int
    numberOfRecordsToCommit = num
}
```

Parameters

Here are the available beCDRMover parameters.

destinationDirectory

Syntax: `destinationDirectory = "dir"`
Description: The destination directory into which EDRs are moved.
Type: String
Optionality: Optional (default used if not set)
Allowed: Default: `/IN/service_packages/E2BE/logs/CDR`
Notes: Must be a valid directory
Example: `destinationDirectory = "/var/edr/UBE/dest"

numberOfRecordsToCommit

Syntax: `numberOfRecordsToCommit = num`
Description: EDRs are moved in batches. This parameter defines the number of records in each batch.
Type: Integer
Optionality: Optional (default used if not set)
Allowed: Default: 10
Notes: Example: `numberOfRecordsToCommit = 10`

oraclePassword

Syntax: `oraclePassword = "str"`
Description: The Oracle password VWS processes use to connect to the E2BE database.
Type: String
Optionality: Optional (default used if not set)
Allowed: Default: e2be_admin
Notes: Example: `oraclePassword = "password"`
oracleService
Syntax: \texttt{oracleService = "name"}
Description: The Oracle service.
Type: String
Optionality: Optional (default used if not set)
Allowed:
Default: 
Notes: 
Example: \texttt{oracleService = ""}

oracleUser
Syntax: \texttt{oracleUser = "name"}
Description: The Oracle user which VWS uses to connect to the E2BE.
Type: String
Optionality: Optional (default used if not set)
Allowed:
Default: \texttt{"e2be\_admin"}
Notes: 
Example: \texttt{oracleUser = "e2be\_admin"}

outDirectory
Syntax: \texttt{outDirectory = "dir"}
Description: The directory from which EDRs are moved.
Type: String
Optionality: Optional (default used if not set)
Allowed:
Default: \texttt{IN/service\_packages/E2BE/logs/CDR}
Notes: 
Example: \texttt{outDirectory = "/var/edr/UBE"}

timeout
Syntax: \texttt{timeout = seconds}
Description: Time (in seconds) before EDR move is regarded as failed.
Type: Integer
Optionality: Optional (default used if not set)
Allowed:
Default: \texttt{10000}
Notes: If this is set to higher than 4, additional "file cannot be deleted" messages will appear in the syslog.
Example: \texttt{timeout = 10000}

Example configuration
This is an example of the beCDRMover section of an eserv.config file on a VWS (comments have been removed).

\begin{verbatim}
beCDRMover = {

description: 'Example configuration',

oracleService = "",
oracleUser = "e2be\_admin",
outDirectory = "/var/edr/UBE",
timeout = 10000
}
\end{verbatim}
raw_text
heartbeatPeriod = microsecs
messageTimeoutSeconds = seconds
maxOutstandingMessages = int
reportPeriodSeconds = seconds
primaryFallbackInterval = seconds
connectionRetryTime = seconds

plugins = [
    {
        config="confStr",
        library="lib",
        function="str"
    }
    [...]
]

confStr = {
    plugin configuration
}

notEndActions = [
    {
        type="str",
        action="[ACK |NACK]"
    }
    [...]
]

billingEngines = [
    {
        id = int,
        primary = { ip="ip", port=port },
        secondary = { ip="ip", port=port }
    }
    [...]
]

Parameters
BeClient has no command line parameters.
The BeClient supports the following parameters from the BE section of eserv.config.
billingEngines
Syntax:     billingEngines = [
            {
                id = int,
                primary = { ip="ip", port=port },
                secondary = { ip="ip", port=port }
            }
            [...]
]
Description:  Overrides connection details that beLocationPlugin (on page 35) obtains from the database.
Type:  Parameter array.
Optionality:  Optional (beLocationPlugin finds connection details if not set).
Allowed:  
Default:  
Notes:  Identifies the Voucher and Wallet Servers and assigns their Internet connection details.
Example:

```json
billingEngines = [
  {
    id = 1,
    primary = { ip="192.0.2.0", port=1500 },
    secondary = { ip="192.0.2.1", port=1500 }
  }
]
```

**id**

**Syntax:**

```
id = int
```

**Description:**

This unique identifier for this Voucher and Wallet Server configuration.

**Type:**

Integer

**Optionality:**

Required, if this section is used

**Allowed:**

None

**Default:**

None

**Notes:**

This parameter is part of the `billingEngines` parameter array.

**Example:**

```
id = 1
```

**primary**

**Syntax:**

```
primary = { ip="ip", port=port }
```

**Description:**

The `primary` parameter group defines the Internet protocol address and associated port number of the primary Voucher and Wallet Server.

**Type:**

Parameter array

**Optionality:**

Mandatory if this section is used

**Allowed:**

None

**Default:**

None

**Notes:**

This parameter is part of the `billingEngines` parameter array.

**Example:**

```
primary = { ip="192.0.2.0", port=1500 }
```

**secondary**

**Syntax:**

```
secondary = { ip="ip", port=port }
```

**Description:**

The `secondary` parameter group defines the Internet protocol address and associated port number of the secondary Voucher and Wallet Server.

**Type:**

Array

**Optionality:**

Required, if this section is used

**Allowed:**

None

**Default:**

None

**Notes:**

This parameter is part of the `billingEngines` parameter array.

**Example:**

```
secondary = { ip="192.0.2.1", port=1500 }
```

**ip**

**Syntax:**

```
ip = "ip"
```

**Description:**

The internet address of the Voucher and Wallet Server.

**Type:**

String

**Optionality:**

Required

**Allowed:**

None

**Default:**

None
Notes: This parameter is part of either the primary or secondary parameter group of the billingEngines parameter array.

Example: ip = "192.0.2.0"

port
Syntax: port = port
Description: The port number associated with the address of the Voucher and Wallet Server.
Type: Integer
Optionality: Required
Allowed: None
Default: None
Notes: This parameter is part of either the primary or secondary parameter group of the billingEngines parameter array.

Example: port = 1500

broadcastOptions
Syntax: broadcastOptions = { 
    aggregateNAckCodes = [config]
}
Description: Name of configuration section for the BeClient Broadcast plug-in libclientBcast.
Type: Parameter array
Optionality: Allowed:
Allowed: 
Default: 
Notes: libclientBcast is used by a range of processes which connect to the beServer, including:
- BeClient
- PlbeClient
- ccsBeOrb

For more information about libclientBcast, see libclientBcast (on page 115).

Example: broadcastOptions = { 
    aggregateNAckCodes = [ ]
}

aggregateNAckCodes
Syntax: aggregateNAckCodes = ["NVOU" ]
Description: When this parameter is set, the BeClient waits for a response from all the VWS pairs in use and filters the responses from the broadcast request using the configured NAck codes.
Type: Parameter array
Optionality: Allowed: NVOU
Default: 


Notes: When a voucher recharge request is broadcast, this ensures that all the available VWS pairs are checked for the required voucher before a voucher not found message is returned to the requesting process.

Example:

**clientName**

Syntax: `clientName = "name"`
Description: The unique client name of the process.
Type: String
Optionality: Required
Allowed: Must be unique.
Default: The host name of the local machine.
Notes: The server generates clientId from a hash of str. If more than one client attempts to connect with the same name, then some connections will be lost.
This parameter is used by libBeClientIF.
Example: `clientName = "scpClient"`

**connectionRetryTime**

Syntax: `connectionRetryTime = seconds`
Description: The maximum number of seconds the client process will wait for a connection to succeed before attempting a new connection.
Type: Integer
Optionality: Required

Default: 5
Notes: This parameter is used by libBeClientIF.
Example: `connectionRetryTime = 2`

**heartbeatPeriod**

Syntax: `heartbeatPeriod = microsecs`
Description: The number of microseconds during which a Voucher and Wallet Server heartbeat message must be detected, or the BeClient process will switch to the other VWS in the pair.
Type: Integer
Optionality: Required
Allowed: 0 disable heartbeat detection. positive integer heartbeat period.
Default: 3000000
Notes: 1 000 000 microseconds = 1 second.
If no heartbeat message is detected during the specified time, client process switches to the other Voucher and Wallet Server in the pair.
This parameter is used by libBeClientIF.
Example: `heartbeatPeriod = 10000000`
maxOutstandingMessages

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

Description: The maximum number of messages allowed to be waiting for a response from the Voucher and Wallet Server.

Type: Integer

Optionality: Required

Allowed: 

Default: If this parameter is not set, the maximum is unlimited.

Notes: If more than this number of messages are waiting for a response from the Voucher and Wallet Server, the client process assumes the Voucher and Wallet Server is overloaded. In this event, the client process refuses to start new calls but continues to service existing calls. The messages are queued until the Voucher and Wallet Server has reduced its outstanding load. This parameter is used by libBeClientIF.

Example: \( \text{maxOutstandingMessages} = 100 \)

messageTimeoutSeconds

Syntax: \( \text{messageTimeoutSeconds} = \text{seconds} \)

Description: The time that the client process will wait for the server to respond to a request.

Type: Integer

Units: Seconds

Optionality: Required

Allowed: 1-604800 Number of seconds to wait.
0 Do not time out.

Default: 2

Notes: After the specified number of seconds, the client process will generate an exception and discard the message associated with the request. This parameter is used by libBeClientIF.

Example: \( \text{messageTimeoutSeconds} = 2 \)

notEndActions

Syntax: \[
\text{notEndActions} = \{
\text{type} = \text{str}, \text{action} = \{\text{ACK|NACK}\}
\}
\]

Description: The notEndActions parameter array is used to define the messages associated with dialogs that should not have their dialog closes, because the dialog is closed by default. This facilitates failover.

Type: Parameter array.

Optionality: Required

Allowed: 

Default:
Notes: If the incoming dialog for a call closes and the last response received was of the notEndActions type, the client process sends an ABRT message. The ABRT message allows the VWS to remove the reservation. An example of this situation would be where slee_acs has stopped working.

This parameter is used by libBeClientIF.

For more information about slee_acs, see ACS Technical Guide.

Example:

```
notEndActions = [
    {type="IR  ", action="ACK "}
    {type="SR  ", action="ACK "}
    {type="SR  ", action="NACK"}
    {type="INER", action="ACK "}
    {type="SNER", action="ACK "}
    {type="SNER", action="NACK"}
]
```

plugins

Syntax:

```
plugins = [
    {
        config=""
        library="lib"
        function="str"
    }
    ...
]
```

Description: Defines any client process plug-ins to run. Also defines the string which maps to their configuration section.

Type: Parameter array

Optionality: Optional (as plug-ins will not be loaded if they are not configured here, this parameter must include any plug-ins which are needed to supply application functions; for more information about which plug-ins to load, see the BeClient section for the application which provides the BeClient plug-ins).

Allowed: Empty (that is, do not load any plug-ins).

Default: Empty (that is, do not load any plug-ins).

Notes: The libclientBcast plug-in must be placed last in the plug-ins configuration list.

For more information about the libclientBcast plug-in, see libclientBcast (on page 115).

This parameter is used by libBeClientIF.

Example:

```
plugins = [
    {
        config="broadcastOptions"
        library="libclientBcast.so"
        function="makeBroadcastPlugin"
    }
]
```

primaryFailbackInterval

Syntax: `primaryFailbackInterval = seconds`

Description: `seconds` defines the failback interval. If the number of seconds since the VWS sent the last request for a session running on the secondary BE is greater than the specified failback interval, then all subsequent requests for the session will be sent to the primary BE. During the failback interval, the secondary BE will synchronize requests to the primary BE.

Type: Integer
Chapter 3, Background Processes

Optionality: Optional (default used if not set)
Allowed:
  - 0 – For immediate failback
  - -1 – To disable primary failback
  - A positive integer
Default: -1
Notes: Setting this parameter will not affect failover behavior. A session will failover to the other BE if a communications error means that it cannot continue processing on the current BE.
Example: primaryFailbackInterval = 10

reportPeriodSeconds
Syntax: reportPeriodSeconds = seconds
Description: The number of seconds separating reports of failed messages.
Type: Integer
Units: Seconds
Optionality: Required
Allowed:
Default: 10
Notes: BeClient issues a failed message report:
  - For timed-out messages
  - For unrequested responses
  - For new calls rejected because of congestion
  - For messages with invalid Voucher and Wallet Server identifiers
  - If new and subsequent requests fail because both Voucher and Wallet Servers have stopped working
  VWS heartbeat detection must be enabled for the parameter to work. Set reportPeriodSeconds to more than heartbeatPeriod.
Example: reportPeriodSeconds = 10

Example configuration
The following configuration is an example BeClient section of eserv.config on a Voucher and Wallet Server node. Comments have been removed.

BeClient = {
  clientName = "scpClient"
  heartbeatPeriod = 3000000
  messageTimeoutSeconds = 2
  maxOutstandingMessages = 100
  reportPeriodSeconds = 10
  primaryFailbackInterval = 10
  connectionRetryTime = 2

  plugins = [
    {
      config="broadcastOptions",
      library="libclientBcast.so",
      function="makeBroadcastPlugin"
    }
  
}
broadcastOptions = {
    aggregateNACKCodes = [
        "NVOU"
    ]
}

notEndActions = [
    {type="IR", action="ACK"},
    {type="SR", action="ACK"},
    {type="SR", action="NACK"},
    {type="INER", action="ACK"},
    {type="SNER", action="ACK"},
    {type="SNER", action="NACK"}
]

billingEngines = [
    {    id = 1,
        primary = { ip="192.0.2.0", port=1500 },
        secondary = { ip="192.0.2.1", port=1500 }
    }
]

Output

The BeClient writes error messages to the system messages file, and also writes additional output to:

/IN/service_packages/CCS/tmp/BeClient.log

Note: The above are defaults and may vary.

beGroveller

Purpose

The beGroveller processes wallets daily on the primary VWS (while the VWS is active) and performs wallet inquiries. This triggers all beVWARs plug-ins that are activated on wallet inquiry (for example, beVWARSExpiry). This activity catches up on due events for wallets that have not been accessed for some time. This keeps the E2BE database relatively up to date and means operations such as MSC deactivation for removed accounts always happen (although later than they are scheduled to occur).

Additionally, the beGroveller executes a night time run to process all the wallets that have not been accessed during the day.

The beGroveller is designed to run on the primary VWS, although it will failover to the secondary if necessary. For more information about which VWS the beGroveller runs on, see beGroveller quorum (on page 130).

Tuning the beGroveller is a balance between the need to keep the database running smoothly for business purposes, and the load imposed by the process. Configuring the beGroveller for less than 100 ms per wallet (= 10 wallets/second) is not recommended.

Process

The beGroveller maintains multiple asynchronous connections to the VWS database; a single connection for each beVWARs requesting grovel activity.
Here is the beGroveller process.

<table>
<thead>
<tr>
<th>Stage</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A connection is assigned to the first beVWARS instance requiring grovel activity and a buffer is opened for it.</td>
</tr>
<tr>
<td>2</td>
<td>The wallets with currently expired buckets are retrieved and stored to the buffer in a collection set for the beVWARS. The buffer is then closed.</td>
</tr>
<tr>
<td>3</td>
<td>The beVWARS requests are then processed directly from the set instead of being continuously fetched from the database.</td>
</tr>
<tr>
<td>4</td>
<td>The beGroveller is responsible for maintaining the set of wallets in the beVWARS buffer. When it is empty, or it drops below a configured threshold, then the buffer is automatically reopened and more wallet details are collected.</td>
</tr>
</tbody>
</table>
| 5     | Successive beVWARS instances perform one of the following:  

- Use an existing connection that is not currently managing a buffer  
- Open a new connection if all the current connections are in operation |
| 6     | At the end of the day the beGroveller creates a list of all the wallets that have not been accessed during the day, and these are processed during the overnight run. |

**Startup**

This task is started by the SLEE, by the following line in `SLEE.cfg`:

```
INTERFACE=beGroveller  beGroveller /IN/service_packages/E2BE/bin instance_count
```

Where `instance_count` is the number of instances to run of the beGroveller process.

**Notes:**

- To enable beGroveller to run, you must set the `enableGrovelling` (on page 35) parameter to `true`.  
- If you configure the SLEE to run multiple instances of the beGroveller, then each beGroveller process will have the value of `instance_count` - 1 appended to the process name. So the master beGroveller process will be named `beGroveller0` and subsequent slave beGroveller processes will be named `beGroveller1`, `beGroveller2` and so on. If you configure only one instance of the beGroveller, then nothing will be appended to the process name.

For more information about configuring SLEE interfaces, see *SLEE Technical Guide*.

**Configuration**

The beGroveller uses parameters from these parameter groups in the `eserv.config` file on VWS nodes:

- `beGroveller`  
- `beVWARS groverl parameters` (on page 93)

beGroveller also uses the `enableGrovelling` (on page 35) shared parameter from the BE section of `eserv.config`.

The `beGroveller` group contains parameters in the structure shown below.

```python
beGroveller = {  
    quorumHost = "host"  
    maxIDsPerResponse = ids  
    retrySeconds = seconds  
    processExpiredBuckets = true|false  
    consecutiveFetch = num
```
Example configuration

This is an example of the beGrokker section of an eserv.config file on a VWS (comments have been removed).

```json
beGrokker = {
    quorumHost = "produsms-cluster"
    maxIDsPerResponse = 160
    retrySeconds = 60
    processExpiredBuckets = true
    noProcessingTimes = [
        { startsAt = "06:00", endsAt = "09:30" }
        { startsAt = "11:30", endsAt = "14:00" }
        { startsAt = "16:00", endsAt = "21:00" }
    ]
    connectionRetryTime = 60
    heartbeatPeriod = 300000000
    filledBufferThreshold = 480
    ludProcessingTime = "14:04"
}
```

Parameters

Parameters of the beGrokker group are listed below.

collectionRetryTime

**Syntax:**
```
collectionRetryTime = seconds
```

**Description:** The number of seconds between attempts to establish a connection to the beServer on the local VWS and the remote VWS in this pair.

**Type:** Integer

**Optionality:** Optional (default used if not set)

**Allowed:**

**Default:** 60

**Notes:** The connection to beServer establishes whether or not the local and remote VWSs are in the running state.

If it fails to make a connection, beGrokker will log an error to the syslog.

For more information about states, see BE States (on page 20).

**Example:**
```
collectionRetryTime = 60
```
Chapter 3, Background Processes

Default: 3000
Notes: This number is hard coded in versions prior to 2.4.0.22.
Example: consecutiveFetch = 3000

filledBufferThreshold
Syntax: filledBufferThreshold = num
Description: Threshold for the minimum number of wallet id entries stored in the buffer. A refill is needed when the number of entries in the buffer falls below this number.
Type: Integer
Optionality: Required
Allowed: Default: 320
Notes: A separate buffer will be used for each beGroveler client. The beGroveler will continue to fetch wallets until all the client buffers are full. It will then wait until the number of entries in one of the buffers falls below the minimum before fetching more wallets.
Example: filledBufferThreshold = 400

heartbeatPeriod
Syntax: heartbeatPeriod = microsecs
Description: The heartbeat period for the beGroveler connection to the beServer through beClientIF.
Type: Integer
Optionality: Optional (default used if not set)
Allowed: Default: 300000000 (300 seconds)
Notes:
Example: heartbeatPeriod = 300000000

ludProcessingTime
Syntax: ludProcessingTime = "HH:MM"
Description: Defines the hour of the day when the last used date logic will execute. If the beGroveler starts at a later time in the day, then the last used date logic processing will be delayed until the next day. If this hour occurs in a no processing period, then the last used date logic processing will be delayed until the end of the no processing period.
Type: String
Optionality: Optional (default used if not set)
Allowed: A valid time in the format HH:MM
Default: 00:00
Notes:
Example: ludProcessingTime = "10:00"

maxIDsPerResponse
Syntax: maxIDsPerResponse = ids
Description: The number of wallet IDs to send to a beVWARS process when it requests wallets
to grovel.

Type: Integer
Optionality: Optional (default used if not set)
Allowed: 160
Default: 160
Notes: beVWARS processes request IDs when they run out other work to do. Setting too low will make groveling slow. Setting it too high will make the response exceed the SLEE event size. 1k events will fit in about 160. 2k events will fit in ~330. Fit in as many as your event size will allow.

For more information about SLEE event sizes, see SLEE Technical Guide.

Example: maxIDsPerResponse = 160

noProcessingTimes
Syntax: noProcessingTimes = [
    {startsAt = "HH:MM", endsAt = "HH:MM"}
    ...
]
Description: The time periods during each day when beGroveller should not return any wallet IDs to beVWARS which are requesting wallet IDs to grovel.
Type: Array
Optionality: Optional (default used if not set)
Allowed: No time restrictions.
Default: No default
Notes: Example: noProcessingTimes = [
    { startsAt = "06:00", endsAt = "09:30" }
    { startsAt = "11:30", endsAt = "14:00" }
    { startsAt = "16:00", endsAt = "21:00" }
]
startsAt
Syntax: startsAt = "HH:MM"
Description: The hour and minute to start a period of not sending wallets to be groveled beVWARS processes.
Type: String
Optionality: Required if noProcessingTimes is set
Allowed: No default
Default: No default
Notes: The period is finished by the endsAt (on page 56) parameter paired with this startsAt parameter in the {} set.
This parameter is part of the noProcessingTimes (on page 56) parameter array.
Example: startsAt = "06:00"
endsAt
Syntax: endsAt = "HH:MM"
Description: The hour and minute to finish a period of not sending wallets to be groveled beVWARS processes.
Type: String
Optionality: Required if noProcessingTimes is set
Allowed: No default
Default: No default
Notes: The period is started by the startsAt (on page 56) parameter paired with this endsAt parameter in the {} set.
This parameter is part of the noProcessingTimes (on page 56) parameter array.
Example: endsAt = "09:30"

processExpiredBuckets
Syntax: processExpiredBuckets = true|false
Description: Activates or deactivates expired bucket processing.
Type: Boolean
Optionality: Optional (default used if not set)
Allowed: true Activate expired bucket processing. The beGroveler will fetch wallets as often as required and process buckets based on the bucket expiry date.
false Deactivate expired bucket processing. The beGroveler will fetch wallets for processing once per day at the time set by the ludProcessingTime parameter.
Default: true
Notes: Example: processExpiredBuckets = true

quorumHost
Syntax: quorumHost = "host"
Description: The host name or IP address of a machine on the same VWS subnet to use as a quorum device. The quorum machine is used to break a tie when trying to decide if beGroveler should allow the beVWARS processes to process wallets.
Type: String
Optionality: Required. If this is not set, or if the specified machine is not on the same VWS subnet, grovelling may not start (see Notes).
Allowed: Not specified
Default: Not specified
Notes: This value is used when a beGroveler cannot see the other VWS in a pair (that is, it cannot ping the other VWS). In this case, beGroveler needs to decide which VWS is partitioned from the rest of the network. If this beGroveler can see quorumHost but not the other VWS, it will grovel. This means that quorumHost must be another device on the same subnet that the VWS nodes use for communication.
A good value for quorumHost may be the logical address of a SMS cluster, or the IP address of a non-clustered SMS, but you should confirm this with the network administrator.
Example: quorumHost = "produsms-cluster"
retrySeconds

Syntax: 
retrySeconds = seconds

Description: How many seconds to tell beVWARS to wait before sending another request for wallet IDs to process. Used when beGroveler cannot find any wallets which need groveling to send to a beVWARS which has requested wallets to grovel.

Type: Integer

Optionality: Optional (default used if not set)

Allowed: 
- 0: beGroveler will attempt to calculate a useful delay to set. Either:
  - the time that a wallet for the requesting beVWARS will expire + 1 minute (up to a maximum of 1 hour), or
  - 300 (five minutes).

Default: 0

Notes:
The number of seconds beVWARS should wait before asking for more wallet IDs to grovel.

Example: retrySeconds = 60

beServer

Purpose

Handles connections from client processes (including BeClient processes) and controls routing to beVWARS processes.

It maintains a list of connected clients, and loads plug-ins to handle different request types.

The beServer is a finite state machine, handling one request at a time until either a response can be sent back, or more information is needed and a further request is sent to the beVWARS.

The beServer deals with:

- Multiple client connections (via be protocol)
- Pluggable message handlers (per message type x message version)
- Call context for call state for plug-ins
- SLEE event message passing
- Switchable accepting messages from client
- Resynchronizable call context.

Plug-ins

beServer can be extended by:

- Routing handlers specified in the messageRoutingPlugins (on page 63) parameter (such as libbeMsgRouterDefault (on page 114))
- Message handlers specified in the handlers (on page 62) parameter.

The beServer will attempt to process messages using its own handlers; if no handler is found the message will be sent to the beVWARS for processing. Message handlers are generally provided by other applications such as CCS to provide application-specific functions such as asking the beVWARS (through the SLEE) for account information, reservations, and billing.

For more information about the plug-ins provided by CCS, see CCS Technical Guide.
About running multiple beServer processes

You can run multiple instances of the beServer to improve performance. The first beServer process (beServer0) is the master beServer and all other instances of the beServer are its slave processes. The master beServer determines which slave beServer to use for each new VWS client connection. It checks the status of the slave beServers and load balances client connections across all slave beServers on the VWS (the master beServer also acts as a slave in this respect and will assign connections to itself as required by the connection loading). Once a VWS client connection has been assigned to a slave beServer, it will remain attached to that slave beServer for the lifetime of the connection.

Note: You can configure the master beServer process to always handle specific VWS client interface connections itself. By default, this includes the beGroveller and ccsMfileCompiler connections. See clientLoadWeightings (on page 60) for more information.

Startup

This task is started by the SLEE, by the following line in SLEE.cfg:

```
INTERFACE=beServer beServerStartup.sh /IN/service_packages/E2BE/bin instance_count EVENT
```

Where `instance_count` is the number of instances to run of the beServer process.

Note: If you configure the SLEE to run multiple instances of the beServer, then each beServer process will have the value of `instance_count - 1` appended to the process name. So the first beServer process will be named beServer0 and subsequent beServer processes will be named beServer1, beServer2 and so on. If you configure only one instance of the beServer, then nothing will be appended to the process name.

For more information about configuring SLEE interfaces, see SLEE Technical Guide.

Configuration

The beServer is configured by the parameters in the following section of eserv.config file:

```plaintext
beServer = {
    clientSelectTime = microsecs
    quiesceLength = microsecs
    serverPortOverride = port
    clientSocketBufferSize = bytes
    maxDownstreamQueueLength = int
    downstreamOverloadSleepUSec = microsecs
    dbConnCheckTime = seconds
    recoveryReportInterval = seconds

    notEndActions = [
        {type="str", action="[ACK|NACK]"}
    ]

    handlers = [
        "lib"
    ]

    messageRoutingPlugins = [
        "lib"
    ]
}```
Chapter 3

```
msgRouterDefault = {
  roundRobinTypes = [
    "TYPE"
  ];
  routeOnVoucherNumber = true|false
}

purge = {
  purgeInterval = seconds
  vvarsTimeout = seconds
  expectedKeep = seconds
  noExpectedKeep = seconds
}

routingVoucherNumberLength = int
slaveLocalSocketDirectory = "directory"

clientLoadWeightings = [
  {name="client_name", weighting=value}
  {name="client_name", weighting=value}
  ...
]
```

Parameters

Parameters of the beServer group are listed below.

clientLoadWeightings

**Syntax:**
```
clientLoadWeightings = [
  {name="client_name", weighting=value}
  {name="client_name", weighting=value}
  ...
]
```

**Description:**
Defines the load weighting value to assign to each type of client connected to the beServer. This improves load sharing over multiple beServer interfaces.
- `client_name` is the name of a client interface configured in SLEE.cfg.
- `value` is the load weighting value and indicates the expected traffic load from the specified client interface. A larger value indicates a greater expected load.

The weighting value for the beGroeveller and ccsMFileCompiler clients should be zero (0) and you should not change their value. Setting the weighting value to zero forces the master beServer to always handle the connection itself.

**Type:** Array

**Optionality:** Optional (default used if not set)

**Allowed:**
- Default: 100 except ccsMfileCompiler (default 0), and beGroeveller (default 0)

**Notes:**
The clientLoadWeightings configuration is used when there are multiple instances of the beServer interface running. It is not used if only one beServer interface is running.
Example:

```python
clientLoadWeightings = [
    {name="ccsBeClient", weighting=200}
    {name="ccsMFileCompiler", weighting=0}
    {name="ccsBeGroveller", weighting=0}
    {name="ccsBeOrb", weighting=10}
    {name="ccsBeResync", weighting=100}
    {name="osaChamScs", weighting=100}
    {name="ccsBatchCharge", weighting=10}
    {name="ccsDomainMigration", weighting=50}
    {name="ccsPeriodicCharge", weighting=100}
    {name="ccsChangeDaemon", weighting=50}
    {name="ccsSLEEChangeDaemon", weighting=50}
    {name="PIbeClient", weighting=10}
]
```

clientSelectTime

**Syntax:**

```plaintext
clientSelectTime = microsecs
```

**Description:**
The number of microseconds between each instant where beServer checks the SLEE for events.

**Type:**
Integer

**Optionality:**
Optional (default used if not set)

**Allowed:**

**Default:** 1000000

**Notes:**
- 1 000 000 microseconds = 1 second.
- If an event is waiting on the SLEE, beServer ignores this setting and makes the next check immediately afterwards. This allows a second event to be detected without delay.
- If the `eserv.config` file is reloaded, beServer will re-read the `clientSelectTime` parameter.

Example:

```plaintext
clientSocketBufferSize = 10240
```

dbConnCheckTime

**Syntax:**

```plaintext
dbConnCheckTime = seconds
```

**Description:**
The number of seconds between each check that beServer is connected to, and logged on to, the Oracle database.

**Type:**
Integer

**Optionality:**
Optional

**Allowed:**

**Default:**

**Notes:**

Example:

```plaintext
clientSocketBufferSize = 10240
```
Default: 1
Notes: If the Oracle database is not available, the current Voucher and Wallet Server is disabled and BeClient routes calls to the other Voucher and Wallet Server.
Example: dbConnCheckTime = 1

downstreamOverloadSleepUSec
Syntax: downstreamOverloadSleepUSec = microsecs
Description: If a downstream process like beVWARS is overloaded, this parameter sets the number of microseconds that beServer will wait before rechecking the process.
Type: Integer
Optionality: Required
Allowed: Default:
Notes: • 1 000 000 microseconds = 1 second.
    • This value must be shorter than the SLEE watchdog timeout period.
Example: downstreamOverloadSleepUSec = 100000

handlers
Syntax: handlers = [
    "lib"
    [...] ]
Description: The handlers parameter array contains plug-in library files that beServer must load.
Type: Parameter array
Optionality: Optional
Allowed: Default:
Notes: • Plug-in library files contain message handlers for requests from clients. A typical file might be libbeServerPingPlugin.so.
    • The order that plug-in files are listed in the array is important. A handler can be loaded twice, causing the last handler to be the one used.
    • If the eserv.config file is reloaded, beServer will re-read the plug-in library files in the handlers parameter array.
Example: handlers = [
    "libbeServerPingPlugin.so"
]

quiesceLength
Syntax: quiesceLength = microsecs
Description: The number of microseconds that the beServer will restrict traffic to only sending responses to outstanding requests from clients.
Type: Integer
Optionality: Optional (default used if not set)
Allowed: Default: 100000 (1/10 second)
Notes: beServer will not read any new work during a quiesced interval so the Voucher and Wallet Server has a chance to confirm the result of requests to the clients.
This minimizes failing over requests to the other VWS in the pair that have been successfully processed on this VWS, but the confirmation has not been sent to the client.

Client requests that have yet to be read will build up during this time, and when they exceed the maximum queue length, BFT will kick in on the client. The beServer will close the socket once quiesceLength has passed, and all traffic will be directed at the other VWS in this pair.

This value should be set to the maximum time it takes to process all outstanding requests currently on the SLEE. Any longer and outstanding requests on the sockets will be delayed unnecessarily.

For more information about VWS error states and recovery, see Process Failure Recovery (on page 129).

Example: quiesceLength = 100000

maxDownStreamQueueLength
Syntax: maxDownStreamQueueLength = len
Description: The maximum number of pending events on any beVWARS.
Type: Integer
Optionality: Required
Allowed: 
Default: 
Notes: If pending events exceed this number, beServer refers to the downstreamOverloadSleepUSEc parameter.
Example: maxDownStreamQueueLength = 1000

messageRoutingPlugins
Syntax: messageRoutingPlugins = [ "lib"
 ]
Description: Which message routing plug-ins to load.
Type: Array
Optionality: Optional (default used if not set)
Allowed: 
Default: 
Notes: These plug-ins tell the beServer which beVWARS to pass requests to. Requests based on a wallet or a voucher must continue to be serviced by the same beVWARS so it can keep the wallet or voucher cached.
For more information, see libbeMsgRouterDefault (on page 114).
Example: messageRoutingPlugins = [ "libbeMsgRouterDefault.so"
 ]

msgRouterDefault
Syntax: msgRouterDefault = {
    roundRobinTypes = []
}
Description: Defaults for the message routing plug-ins loaded by messageRoutingPlugins (on page 63).
Type: Array
Optionality: Allowed
Default:
Notes: Includes the roundRobinTypes (on page 64) parameter
Example:

roundRobinTypes
Syntax: 
    roundRobinTypes = [
        "type"
    ]
Description: Default routing for libbeMsgRouterDefault.so.
Type: Array of four-character strings.
Optionality: Optional (default used if not set)
Allowed:
Default:
Notes: If a message does not have a [WALT] or [VNUM] tag and its message type is in this array, it will be round robined around beVWARS to share load. The CCS VI message may or may not have a [VNUM] field. You can also organize the elements in this array in one line, using a comma ',' to separate the types.
Example: 
    roundRobinTypes = [
        "VI"
    ]

routeOnVoucherNumber
Syntax: 
    routeOnVoucherNumber = true|false
Description: What method to use to determine which beVWARS process to route voucher redeem requests to.
Type: Boolean
Optionality: Optional (default used if not set)
Allowed: true Use a hash of the Voucher Number to route to beVWARS.
        - Compatible with CCS 3.1.4 and earlier.
        false Use Voucher ID to route to beVWARS.
        - Compatible with CCS 3.1.5 and later.
Default: true
Notes: This parameter is used by the libbeMsgRouterDefault library.
Example: routeOnVoucherNumber = false

notEndActions
Syntax: 
    notEndActions = [
        {type="type", action="ACK|NACK"}
    ]
Description: This parameter array identifies messages that will be followed by subsequent message.
Type: Parameter array
Chapter 3, Background Processes

Optionality: Required

Allowed: 
Default: 
Notes: 
Example: 

```
notEndActions = [
    {type="IR ", action="ACK "}
    {type="SR ", action="ACK "}
    {type="SR ", action="NACK"}
    {type="INER", action="ACK "}
    {type="SNER", action="ACK "}
    {type="SNER", action="NACK"}
]
```

purge

Syntax: 
```
purge = {
    purgeInterval = seconds
    vwarsTimeout = seconds
    expectedKeep = seconds
    noExpectedKeep = seconds
}
```

Description: The purge parameter group contains parameters that control purges.

Type: Parameter group.

Optionality: Optional

Allowed: 
Default: 
Notes: Running purge stresses the system with high loads.

Example:

```
expectedKeep
```

Syntax: 
```
expectedKeep = seconds
```

Description: A plug-in can specify the number of seconds it will wait for a request for a context that it wants to keep. This parameter sets additional time, after the plug-in's time, that beServer keeps a context if (during this period) no request for the context is made.

Type: Integer

Optionality: Optional

Allowed: 
Default: 60

Notes: 
- This parameter is part of the purge parameter group.
- If the eserv.config file is reloaded, beServer will re-read the parameter.

Example: 
```
expectedKeep = 60
```

noExpectedKeep

Syntax: 
```
oExpectedKeep = seconds
```

Description: If the plug-in does not specify a time it will wait for a request for a wanted context, this parameter defines the number of seconds that beServer will keep the context.

Type: Integer

Optionality: Optional

Allowed: 
Default: 
Notes: 
Example:
purgeInterval
Syntax: purgeInterval = seconds
Description: The number of seconds between purges.
Type: Integer
Optionality: Optional
Allowed: Default: 300
Notes: 
  - This parameter is part of the purge parameter group.
  - If the eserv.config file is reloaded, beServer will re-read the parameter.
Example: purgeInterval = 300

vwarsTimeout
Syntax: vwarnsTimeout = seconds
Description: The number of seconds between the moment that beServer sends a request to the beVWARS and the moment that beServer fabricates an exception response.
Type: Integer
Units: Seconds
Optionality: Optional
Allowed: Default: 10
Notes: 
  - This parameter is read during a purge.
  - This parameter is part of the purge parameter group.
  - If the eserv.config file is reloaded, beServer will re-read the parameter.
Example: vwarnsTimeout = 10

recoveryReportInterval
Syntax: recoveryReportInterval = seconds
Description: The number of seconds between logging each recovery report to the syslog while in recovery mode.
Type: Integer
Optionality: Optional (default used if not set)
Allowed: Default: 60
Notes: The recovery report records how many beVWARS processes beServer is waiting to go into running state before it will go into running state.
For more information about the different states, see Process Failure Recovery (on page 129).
Example: recoveryReportInterval = 60
Chapter 3

routingVoucherNumberLength

Syntax: routingVoucherNumberLength = len

Description: The length of the prefix of the voucher number to use for routing voucher messages to beVWARS processes.

Type: Integer

Optionality: Optional (default used if not set)

Allowed:

Default: 10

Notes: For CCS vouchers, this should match the length of the voucher number, not the length of the voucher signature. For more information about voucher numbers and voucher signatures, see CCS Voucher Management User’s Guide.

Example: routingVoucherNumberLength = 10

serverPortOverride

Syntax: serverPortOverride = port

Description: The port number beServer uses as an alternative to the one defined by the beLocationPlugin (on page 35).

Type: Integer

Optionality: Optional

Allowed:

-1 Do not override beLocationPlugin.

any valid port Port for beServer to use.

Default: -1

Notes: This parameter is usually used for testing.

Example: serverPortOverride = 1500

slaveLocalSocketDirectory

Syntax: slaveLocalSocketDirectory = "directory_name"

Description: Specifies the directory to use for files created by interprocess communication (IPC) objects, such as sockets, semaphores, and shared memory. The IPC objects enable communication between master and slave beServer interfaces.

Type: String

Optionality: Optional (default used if not set)

Allowed: A valid directory location.

Default: /tmp

Notes: None

Example: slaveLocalSocketDirectory = "/tmp"

Example configuration

This is an example of the beServer section of the eserv.config file on a VWS node (comments have been removed).

beServer = {
    clientSelectTime = 1000000
    quiesceLength = 100000
    serverPortOverride = 1500
    clientSocketBufferSize = 10240
    maxDownstreamQueueLength = 1000
    downstreamOverloadSleepUSec = 100000
}
dbConnCheckTime = 5
recoveryReportInterval = 60

notEndActions = [
    {type="IR", action="ACK"},
    {type="SR", action="ACK"},
    {type="SR", action="NACK"},
    {type="INER", action="ACK"},
    {type="SNER", action="ACK"},
    {type="SNER", action="NACK"}
]

handlers = [
    "libbeServerPingPlugin.so"
]

messageRoutingPlugins = [
    "libbeMsgRouterDefault.so"
]

msgRouterDefault = {
    roundRobinTypes = [
        "VI"
    ]
    routeOnVoucherNumber = true
}

purge = {
    purgeInterval = 300
    vWarsTimeout = 10
    expectedKeep = 60
    noExpectedKeep = 3600
}

routingVoucherNumberLength = 10
slaveLocalSocketDirectory = "/tmp"

clientLoadWeightings = [
    {name="ccsBeClient", weighting=200},
    {name="ccsMFileCompiler", weighting=0},
    {name="ccsBeGroveller", weighting=0},
    {name="ccsBeOrb", weighting=10},
    {name="ccsBeResync", weighting=100},
    {name="osaChamScs", weighting=100},
    {name="ccsBatchCharge", weighting=10},
    {name="ccsDomainMigration", weighting=50},
    {name="ccsAccount", weighting=10},
    {name="ccsPeriodicCharge", weighting=100},
    {name="ccsChangeDaemon", weighting=50},
    {name="ccsSLEEChangeDaemon", weighting=50},
    {name="PibeClient", weighting=10}
]

} # BE.beServer

Output

The beServer writes error messages to the system messages file, and also writes additional output to the following location by default:

/IN/service_packages/E2BE/tmp/beServer.log
beSync

Purpose

Synchronizes data between the Voucher and Wallet Servers in a VWS pair.

beSync collects all updates and reservations being made, and writes them to disk. It then reads them from disk and sends them to the other VWS, as and when it can.

For more information on beSync and how it interacts with other VWS components, see *Synchronization* (on page 14).

Startup

This task is started by the SLEE, by the following line in *SLEE.cfg*:

```plaintext
INTERFACE=beSync   beSyncStartup.sh   /IN/service_packages/E2BE/bin instance_count
```

Where `instance_count` is the number of instances to run of the beSync process.

**Note:** If you configure the SLEE to run multiple instances of the beSync process, then each beSync process will have the value of `instance_count` - 1 appended to the process name. So the first beSync process will be named beSync0 and subsequent beSync processes will be named beSync1, beSync2 and so on. If you configure only one instance of beSync then nothing is appended to the process name.

For more information about configuring SLEE interfaces, see *SLEE Technical Guide*.

Configuration

beSync accepts the following parameters from *eserv.config*.

```plaintext
beSync = {
    shared = {
        noWorkSleepTime = seconds
        spoolDirectory = "dir"
        spoolChunkSize = num
        badFileDirectory = "dir"
        maxDownstreamQueueLength = num
        downstreamOverloadSleepUSec = int
    }
    sink = {
        inSyncThresholdSeconds = seconds
        inSyncReportingPeriodRecords = seconds
        remoteBEhostname="host"
        remoteBEport=port
        retryConnectionDelaySeconds = seconds
        maxRetriesBeforeStart = num
        localUpdateChunkSize = size
        heartbeatPeriodSeconds = seconds
    }
    source = {
        listenInterface="ip"
    }
}```
listenPort = port
recordSendingChunkSize = num
maxQueueLength = num

Parameters

Here are the parameters in the beSync section.

shared parameters
The shared sub-section of beSync defines the beSync shared items.

badFileDirectory
Syntax: badFileDirectory = "dir"
Description: Directory to move corrupted resync files to.
Type: String
Optionality: Optional (default used if not set)
Allowed: Any directory path.
Default: "/IN/service_packages/E2BE/tmp"
Notes: Files in this directory will be called file.bad.
Example: badFileDirectory = "/IN/service_packages/E2BE/tmp"

downstreamOverloadSleepUSec
Syntax: downstreamOverloadSleepUSec = int
Description: When a downstream process, a beVWARS is overloaded, sleep for this period before rechecking.
Type: Integer
Optionality: Optional (default used if not set)
Allowed: 
Default: 100000
Notes: This value must be shorter than the SLEE watchdog timeout period.
Example: downstreamOverloadSleepUSec = 100000

maxDownstreamQueueLength
Syntax: maxDownstreamQueueLength = num
Description: The maximum number of pending events on any beVWARS. When more than this number of events are queued on any of the processes, beSync will sleep.
Type: Integer
Optionality: Optional (default used if not set)
Allowed: 
Default: 10000
Notes: See maxQueueLength (on page 74) for throttling based on the remote beSync.

Example: maxDownstreamQueueLength = 10000

Important: Care should be taken when setting BE.beSync.maxDownstreamQueueLength as this is the value that slows a full resync by keeping each of the beVWARS processes busy. If you allow a full resync to run as fast as possible, it will use up all of the events.
noWorkSleepTime
Syntax: \texttt{noWorkSleepTime = seconds}
Description: The sleep time in seconds.
Type: Integer
Optionality: Optional (default used if not set)
Allowed: 
Default: 0.2
Notes: Should be small (0.x) in production, and larger in test (2.0). Lower values will cause more CPU usage.
Example: noWorkSleepTime = 0.2

spoolChunkSize
Syntax: \texttt{spoolChunkSize = num}
Description: The number of records to read and send in one cycle.
Type: Integer
Optionality: Optional (default used if not set)
Allowed: 
Default: 16
Notes: 
Example: spoolChunkSize = 16

spoolDirectory
Syntax: \texttt{spoolDirectory = "dir"}
Description: This is where all transactions are written to disk by beVWARS so they can be replayed to one of the following:
\begin{itemize}
  \item The remote VWS in the pair
  \item If there has been a failure, the local VWS in the pair
\end{itemize}
Type: String
Optionality: Optional (default used if not set)
Allowed: 
Default: /IN/service_packages/E2BE/sync
Notes: Available space in the directory set by this parameter is checked by the beSync against the limits set in the Disk space parameters.
Example: spoolDirectory = "/var/logs/sync"

sink parameters
The \texttt{sink} sub-section of beSync defines the sink parameters for beSync. This is the component that receives operations from the remote beSync.

heartbeatPeriodSeconds
Syntax: \texttt{heartbeatPeriodSeconds = seconds}
Description: How often in seconds heartbeat packets are sent on a connection.
Type: Integer
Optionality: Optional (default used if not set)
Allowed: 
Default: 10
Notes: Example: \( \text{heartbeatPeriodSeconds} = 10 \)

\( \text{inSyncThresholdSeconds} \)

Syntax: \( \text{inSyncThresholdSeconds} = \text{seconds} \)
Description: How close (in seconds) to real-time before the beSync is enabled.
Type: Integer
Optionality: Optional (default used if not set)
Allowed: Default: 5
Notes: Example: \( \text{inSyncThresholdSeconds} = 5 \)

\( \text{inSyncReportingPeriodRecords} \)

Syntax: \( \text{inSyncReportingPeriodRecords} = \text{seconds} \)
Description: The number of seconds between checks against real-time.
Type: Integer
Optionality: Optional (default used if not set)
Allowed: Default: 10000
Notes: Example: \( \text{inSyncReportingPeriodRecords} = 10000 \)

\( \text{localUpdateChunkSize} \)

Syntax: \( \text{localUpdateChunkSize} = \text{size} \)
Description: Tuning parameter.
Type: Integer
Optionality: Optional (default used if not set)
Allowed: Default: 100
Notes: Example: \( \text{localUpdateChunkSize} = 100 \)

\( \text{maxRetriesBeforeSeconds} \)

Syntax: \( \text{maxRetriesBeforeSeconds} = \text{num} \)
Description: The number of attempts to contact the other VWS in the pair before we start regardless.
Type: Integer
Optionality: Optional (default used if not set)
Allowed: Default: 5
Notes: Example: \( \text{maxRetriesBeforeSeconds} = 5 \)
remoteBEhostname
Syntax: \( \text{remoteBEhostname} \) = "host"
Description: Overrides the DB configuration of the remote VWS host.
Type: String
Optionality: Optional (default used if not set)
Allowed: 
Default: The name of the remote BE host.
Notes:
Example:

remoteBEport
Syntax: \( \text{remoteBEport} \) = \text{port}
Description: Overrides the DB configuration of the remote VWS port
Type: Integer
Optionality: Optional (default used if not set)
Allowed: Valid port number
Default: 2001
Notes:
Example: \( \text{remoteBEport} = 2001 \)

retryConnectionDelaySeconds
Syntax: \( \text{retryConnectionDelaySeconds} \) = \text{seconds}
Description: The maximum number of seconds between connection attempts
Type: Integer
Optionality: Optional (default used if not set)
Allowed: 
Default: 30
Notes:
Example: \( \text{retryConnectionDelaySeconds} = 30 \)

source parameters
The source sub-section `beSync` defines the source parameters for `beSync`. This is the component that sends operations to the remote `beSync`.

listenInterface
Syntax: \( \text{listenInterface} \) = "ip"
Description: Overrides the DB configuration for what we listen to
Type: String
Optionality: Optional (default used if not set)
Allowed: Valid IP address
Default: 0.0.0.0
Notes:
Example: \( \text{listenInterface} = "0.0.0.0" \)
listenPort

Syntax: \textit{listenPort = port}
Description: Overrides the DB configuration
Type: Integer
Optionality: Optional (default used if not set)
Allowed: Default: 2001
Notes: Example: \textit{listenPort = 2001}

maxQueueLength

Syntax: \textit{maxQueueLength = num}
Description: How many messages can queue on the socket before we stop sending and stop getting work from the beVWARS
Type: Integer
Optionality: Optional (default used if not set)
Allowed: Default: 50
Notes: Example: \textit{maxQueueLength = 50}

recordSendingChunkSize

Syntax: \textit{recordSendingChunkSize = num}
Description: The number of records to send in one poll cycle
Type: Integer
Optionality: Optional (default used if not set)
Allowed: Default: 50
Notes: Example: \textit{recordSendingChunkSize = 50}

Example configuration

This is an example of the \texttt{beSync} section of the \texttt{eserv.config} file on a VWS (comments have been removed).

\begin{verbatim}
beSync = {
  shared = {
    noWorkSleepTime = 0.2
    spoolDirectory = "/IN/service_packages/E2BE-sync"
    spoolChunkSize = 16
    badFileDirectory = "/IN/service_packages/E2BE/tmp"
    maxDownstreamQueueLength = 10000
    downstreamOverloadSleepUSec = 100000
  }

  sink = {
    inSyncThresholdSeconds = 5
    inSyncReportingPeriodRecords = 10000
  }
\end{verbatim}
Output

The beSync writes error messages to the system messages file, and also writes additional output to:

/IN/service_packages/E2BE/tmp/beSync.log

Note: The above are defaults and may vary.

beServiceTrigger

Purpose

beServiceTrigger sends BPL requests to instances of the xmlTcapIF and OSD requests to the osdInterface running on separate SLC nodes within the same IN platform. It runs as a SLEE interface on the primary VWS only.

beServiceTrigger accepts beServiceTrigger events from other BE SLEE interfaces running on the same VWS. For each beServiceTrigger event received, it first checks whether an operationSetName is defined in the event. If an operationSetName is:

- Defined, it sends the related OSD operation to the OSD interface running on a separate SLC node
- Not defined, it creates and sends a new BPL request to an available instance of an XML TCAP interface running on a separate SLC node.

Example

An example of the use of the beServiceTrigger is the Rewards plug-in in the beVWARS. The Rewards plug-in sends a beServiceTrigger request to execute a control plan to apply rewards to subscribers on non-VWS charging domains.

Characteristics

beServiceTrigger has the following characteristics:

- It only accepts beServiceTrigger events sent by other SLEE interfaces running on the same (primary) VWS
- When processing beServiceTrigger events, a new BPL or OSD request is issued to the next available XML TCAP or OSD interface without any acknowledgment to the requesting interface. The traffic between the beServiceTrigger and each particular xmlTcapIf / osdInterface is handled synchronously. No BPL / OSD request will be sent to the same xmlTcapIf/ osdInterface instance until the processing of the previous BPL / OSD request has finished.
- Apart from processing incoming beServiceTrigger events, the beServiceTrigger interface communicates with the beVWARS interfaces in order to produce EDRs as a result of processing BPL / OSD responses.
• Overall, the processing of beServiceTrigger events and connections to different xmlTcapIf / osdInterface instance is done asynchronously. This allows events to be processed and requests to different xmlTcapIf / osdInterface instances to be handled in parallel.

Process

This section describes how beServiceTrigger processes a beServiceTrigger event from the Rewards plug-in of beVWARS.

<table>
<thead>
<tr>
<th>Stage</th>
<th>Description</th>
</tr>
</thead>
</table>
| 1     | beServiceTrigger receives a beServiceTrigger event from the Rewards plug-in and immediately creates a new BPL request ready to be sent to the next available xmlTcapIf.  
**Note:** The Rewards plug-in is not notified about the events received or the BPL requests being sent. |
| 2     | When an xmlTcapIf becomes available, the enqueued BPL request is sent and the xmlTcapIf then becomes unavailable until a response is received or the request times out. |
| 3     | Incoming beServiceTrigger events and BPL responses are handled asynchronously to allow new BPL requests to be sent to available xmlTcapIf instances. |
| 4     | Once a BPL response arrives, the corresponding xmlTcapIf becomes available to process further requests. A request to create a Control Plan Service Invoke EDR (type 7) is sent to the appropriate beVWARS interface. |

Startup

This task is started by the following line in SLEE.cfg:

```
INTERFACE=beServiceTrigger  beSerTrigStartup.sh /IN/service_packages/E2BE/bin EVENT
```

**Note:** Only one instance of the beServiceTrigger interface is allowed per VWS SLEE.

Valid interfaces

The beServiceTrigger requires and uses slightly different configuration depending on the interface used. xmlTcap is the default interface unless the operation set is defined in the request, in which case the OSD interface is used.

XmlTcap parameters

- Control_Plan
- Service_Handle
- scps

OSD parameters

- CCSNamespace
- Username
- Password
- osd_scps
- operationSet
- operation

Parameters common to both intrefaces
XMLTcap Parameters

beServiceTrigger/XMLTcap is configured by the following parameters from the triggering section in the eserv.config file on the VWS:

```plaintext
triggering = {
    Control_Plan = "cpname"
    Service_Handle = "handle"
    scps = [ "ip:port" ]
}
```

**Control_Plan**

- **Syntax:** Control_Plan = "cpname"
- **Description:** The default control plan name that will be used in BPL requests if none is present in the SLEE event.
- **Type:** String
- **Optionality:** Optional
- **Allowed:**
- **Default:** Empty
- **Notes:**
- **Example:** Control_Plan = "Reward"

**scps**

- **Syntax:** scps = [ "ip:port" ]
- **Description:** List of each xmlTcapInterface SLCs that beServiceTrigger connects to.
- **Type:** Array
- **Optionality:** Required
- **Allowed:**
  - **ip** Any IP address in either dotted decimal or symbolic name format.
  - **port** Integer in the range 0 to 65535.
- **Default:** port defaults to 3072
- **Notes:**
  - An example of an Internet protocol address in dotted-decimal format is 192.168.1.56. An example of an address in symbolic name format is primary_smce.
Example:
```
scps = [
    "192.168.1.5:3005",
    primary_smc
]
```

Service Handle

**Syntax:**
```
Service_Handle = "handle"
```

**Description:**
The default service handle that will be used in BPL requests if none is present in the SLEE event.

**Type:** String

**Optionality:** Optional

**Allowed:**

**Default:** Empty

**Notes:**

**Example:**
```
Service_Handle = "CCS_BPL"
```

**OSD Parameters**

`beServiceTrigger/OSD` is configured by the following parameters from the triggering section in the `eserv.config` file on the VWS:

```
triggering = {
    CCSNamespace = "URL",
    Username = "name",
    Password = "password",
    osd_scps = [ "ip:port" ],
    operationSet = "name",
    operation = "name"
}
```

**CCSNamespace**

**Syntax:**
```
CCSNamespace = "URL"
```

**Description:**
The default Namespace that will be put into OSD requests if none is present in the SLEE event.

**Type:** String

**Optionality:** Optional

**Allowed:**

**Default:** Empty

**Notes:**

**Example:**
```
CCSNamespace = "http://eng-prf-zone01-z1/wsdls/ON/CCSNotifications.wsdl"
```

**Password**

**Syntax:**
```
Password = "password"
```

**Description:**
The common http password that will be used in all OSD requests

**Type:** String

**Optionality:** Required

**Allowed:**

**Default:** ""

**Notes:**

**Example:**
```
Password = "password"
```
osd_scps
Syntax: osd_scps = [  "ip:port"  
  ...  
]
Description: List of each xmlTcapInterface SLCs that beServiceTrigger connects to.  Type: Array
Optionality: Required
In any row of the array, the ip must be specified but port is optional.
Allowed:  ip Any IP address in either dotted decimal or
  symbolic name format.
  port Integer in the range 0 to 65535.
Default:  port defaults to 3072
Notes: An example of an Internet protocol address in dotted-decimal format is
  192.168.1.56. An example of an address in symbolic name format is
  cmxdevscp1.
Example:  osd_scps = [  "cmxdevscp1:3072", "cmxdevscp2:3072"  ]

operation
Syntax: operation = "name"
Description: The name of the OSD operation to invoke the service when none is set in the
  SLEE event.
Type: String
Optionality: Optional (default used if not set)
Allowed:  
Default: Empty
Notes:  
Example:  operation = "Notification"

operationSet
Syntax: operationSet = "name"
Description: The name of the operation set that contains the template used to invoke the
  service when none is set in the SLEE event.
Type: String
Optionality: Optional (default used if not set)
Allowed:  
Default: Empty
Notes:  
Example:  operationSet = "NotificationSet"

Username
Syntax: Username = "name"
Description: The common http user name that will be put in all OSD requests
Type: String
Common parameters

beServiceTrigger common parameters are configured by the following from the triggering section in the eserv.config file on the VWS:

```
triggering = {
    edr = true | false
    failureRetryTime = seconds
    storageInterface = "name"
    triggerInterface = "name"
    responseTag = name
    maxRatePerUAS = num
    throttleLife = seconds
    timeBetweenThrottles = millisecs
    maxConnections = integer
}
```

edr
Syntax: \( edr = true | false \)
Description: Should an EDR be produced when a response is received.
Type: Boolean
Optionality: Optional (default used if not set)
Allowed: true, false
Default: false
Notes: Example: \( edr = false \)

failureRetryTime
Syntax: \( failureRetryTime = seconds \)
Description: The length of time in seconds between attempts to send the message to the SLC.
Type: Integer
Optionality: Optional (default used if not set)
Allowed: 
Default: 60
Notes: Example: \( failureRetryTime = 60 \)

maxConnections
Syntax: \( maxConnections = integer \)
Description: The maximum number of connections from beServiceTrigger to interfaces on the SLC.
Type: Integer
Optionality: Optional (default used if not set)
Allowed: 0 or any positive integer. 0 indicates no maximum.
Default: 25
Notes: Increase the value of maxConnections as the number of rows in the be_event_storage table increases.
Example: maxConnections = 25

maxRatePerUAS

Syntax: maxRatePerUAS = num
Description: The maximum rate (messages/second) each SLC is able to handle before throttling.
Type: Integer
Optionality: Optional (default used if not set)
Allowed: Default: 0
Notes: 0 means unlimited.
Example: maxRatePerUAS = 0

responseTag

Syntax: responseTag = name
Description: The name of the tag in the response message to use to populate the EDR.
Type: String
Optionality: Optional (default used if not set)
Allowed: Default: Result
Notes:
Example: responseTag = Result

storageInterface

Syntax: storageInterface = "name"
Description: The name of the interface used to store events for sending later, either due to a failure, or a request for a delayed send.
Type: String
Optionality: Optional (default used if not set)
Allowed: Default: beEventStorageIF
Notes:
Example: storageInterface = "beEventStorageIF"

throttleLife

Syntax: throttleLife = seconds
Description: The length of time in seconds a throttle will exist for before the attempts to back it off.
Type: Integer
Optionality: Optional (default used if not set)
Allowed: Default: 30
Notes:
Example: \[ \text{throttleLife} = 30 \]

**timeBetweenThrottles**

**Syntax:** \[ \text{timeBetweenThrottles} = \text{millisecs} \]

**Description:** The length of time in milliseconds between throttle messages being sent to the storage interface.

**Type:** Integer

**Optionality:** Optional (default used if not set)

**Allowed:**

**Default:** 10

**Notes:**

**Example:** \[ \text{timeBetweenThrottles} = 10 \]

**triggerInterface**

**Syntax:** \[ \text{triggerInterface} = "\text{name}" \]

**Description:** The name of the triggering interface itself.

**Type:** String

**Optionality:** Optional (default used if not set)

**Allowed:**

**Default:** beServiceTrigger

**Notes:**

**Example:** \[ \text{triggerInterface} = "\text{beServiceTrigger}" \]

**Output**

beServiceTrigger writes error messages to the system messages file, and also writes additional output to the location indicated in the startup script, which will usually be set to:

/IN/service_packages/E2BE/tmp/beServiceTrigger.log

**Notification requests**

The VWS directs all notification requests to a SLC OSD interface through the beServiceTrigger (flow 2 in diagram).

**Notification overview**

The OSD interface triggers ACS, which loads a control plan containing the notification node in order to perform delivery (flow 6 in diagram).

If no OSD nodes are available, are unresponsive, or the notification has a Time of Day associated, then it will be stored locally for subsequent delivery.

**Note:** The Wallet Information will be omitted from this, as it will be stale before the notification is sent.

The Time Daemon will poll the Time Dependant notifications stored on the VWSs and trigger OSD requests according to the time. These requests will be throttled and load balanced in order to not overload SLC nodes with large numbers of Control Plan requests.

The VWSs will operate in isolation within the pair, therefore if notifications are delayed and stored on the Primary VWS, they will not be synchronized to the secondary.
If there is a failure or outage in the primary, notifications to be generated will be stored locally on the secondary during the outage. Once the primary is operational again it will process all relevant notifications, while the secondary retains responsibility for notifications generated during the primary outage.
Notification flows

This diagram shows the various notification flows across the NCC platform.

**Flow 1**
The beVWARS plug-ins send SMS information to the beServiceTrigger.
Flow 2
Notification XML messages from the beServiceTrigger to the OSD interface on the SLC.

Flow 3
It is possible that a notification cannot be delivered immediately, either because it has an associated
time period when it may be delivered, or because the delivery attempt failed. In these cases, persistent
storage of these notifications is provided for by a database table.

Flow 4
When the time notification daemon examines the notification entries in the database, it retrieves the
notifications that can now be sent either because their allowable delivery time has been met or because
it is a message retry.

Flow 5
The time notification daemon deletes the active entries from the database and sends delivery request
messages to the beServiceTrigger for each of the active entries.

Flow 6
The OSD interface triggers ACS, which will load a control plan containing the Notification node in order
to perform delivery.

Flow 7
Selection of template by Notification feature node.
The template retrieved is based upon the:
- Language ID
- Template ID
- Customer ID

Flow 8
Notification feature node delivering USSD notification through TCAP interface.
Once the notification node has performed all the parameter substitutions, an internal message shall be
sent through the USSD push action handler to the TCAP interface if the message class is "USSD push"

Flow 9
Chassis action to construct message from template.

Flow 10
Other send message nodes use new chassis action to deliver notifications using Messaging Manager.

beVWARS

Purpose
beVWARS caches and holds the state of all wallets and their associated reservations, and vouchers and
their associated reservations.
The database cannot be relied upon to reflect the state of the running system, as updates are almost
always pending in the Writer. To maintain a single consistent view of the state of an individual record in
the database, all wallet or voucher accesses must come through a beVWARS instance. beVWARS is
responsible for all updates to database field changed during a resync.
beVWARS also handles COMMITing the database updates and writing EDRs. For more information
about this, see Database commits by beVWARS (see "beVWARS data updates" on page 12).
UUC Activation

In order to activate the UUC features, the ccsVWARSReservationExpiry.so entry must be added to the beVWARS plugins section in the /IN/service_packages/eserv.config file “ON THE PRIMARY BE ONLY”. The installation instructions tell you WHEN to do this.

Plug-ins

beVWARS can be extended by:

- Event handlers specified in plugins (on page 90) parameter
- Message handlers specified in the handlers (on page 88) parameter

VWS provides a set of standard beVWARS plug-ins to handle standard interactions. These include beVWARSMergeBuckets (on page 107).

Other handlers are provided by other applications such as CCS to provide application-specific functions such as named event charges. For more information about the plug-ins provided by CCS, see CCS Technical Guide.

Structure of plugins array

Here is the high level structure of the plugins section within the BeVWARS section of the configuration.

```plaintext
BE = {
  beVWARS = {
    plugins = [
      # ... Existing plug-ins here ...
      "ccsVWARSReservationExpiry.so"  # <-- New plugin entry
    ]
  }
}
```

Insert the line as shown in the example above. This plug-in is required to handle the reservation expiry features delivered by UUC.

Note: This line should only be added on the Primary BE. This is because in normal operation, a reservation will expire at exactly the same time on both VWSs. Installing the ccsVWARSReservationExpiry plug-in on the Secondary BE will cause the user to be double-charged.

Startup

This task is started by the SLEE, by the following line in SLEE.cfg:

```plaintext
INTERFACE=beVWARS beVWARSStartup.sh /IN/service_packages/E2BE/bin instance_count EVENT
```

Where `instance_count` is the number of instances to run of the beVWARS process.

Note: beVWARS usually uses more than one beVWARS process. Each beVWARS process will have the value of `instance_count - 1` appended to the process name. So the first beVWARS process will be beVWARS0, and subsequent beVWARS processes will be named beVWARS1, beVWARS2 and so on. If you configure only one instance of the beVWARS, then nothing will be appended to the process name.

For more information about configuring SLEE interfaces, see SLEE Technical Guide.

Configuration

beVWARS accepts the following parameters from eserv.config.

```plaintext
beVWARS = {
  voucherReservationPeriodSeconds = seconds
}
```
useTimeFromClient = true|false
maxTransactionsPerSet = num
maxOpenDialogTime = seconds
maxDownstreamQueueLength = num
downstreamOverloadSleepUSec = microsecs
minResyncReservationLength = seconds
createBucketExpiryDays = days
maxSendReservationsToSync = num
reservationExpiryCheckMilliseconds = millisecs

walletCache = {
  maxSize = MB
  maxLoopSize = num
  checkBeforeFlush = true|false
}

voucherCache = {
  maxSize = MB
  maxLoopSize = num
  checkBeforeFlush = true|false
}

groveller = {
  periodMsec = millisecs
  requestHighWaterMark = num
  walletLowWaterMark = num
  requestTimeout = seconds
  peerDatabaseLogin = "login"
  peerWalletCheckRetrySeconds = seconds
  secondaryConnectionDelaySeconds = seconds
}

duplicateDetection = {
  keepDirectSeconds = seconds
  keepSyncSeconds = seconds
  directMaxDelaySeconds = seconds
  syncMaxDelaySeconds = seconds
}

setLastActivationDateStates = [states]

plugins = ["lib"
  [...]
]

handlers = ["lib"
  [...]
]

syncWriter = {
  maxRecordsPerFile = num
  maxSecondsPerFile = seconds
}

dbWriter = {
  flushPeriod = seconds
  cdrOutputDirectory = "dir"
  balanceCreateBufferSize = num
  balanceUpdateBufferSize = num
  balanceDeleteBufferSize = num
bucketCreateBufferSize = num
bucketUpdateBufferSize = num
bucketDeleteBufferSize = num
walletCreateBufferSize = num
walletUpdateBufferSize = num
walletDeleteBufferSize = num
voucherCreateBufferSize = num
voucherUpdateBufferSize = num
voucherDeleteBufferSize = num

Parameters

Here are the available parameters in the beVWARS section of the eserv.config.

createBucketExpiryDays

Syntax: createBucketExpiryDays = days

Description: In the rare case that the beVWARS must spontaneously create a bucket to preserve the wallet's last use date (that is, when the last bucket has been deleted or a call has been made when the wallet contains no buckets at all). This parameter defines the expiry date to set.

Type: Integer
Units: Days
Optionality: Optional (default used if not set).
Allowed:

- 0 Create the bucket with no expiry date
- positive integer Set the expiry date to this many days in the future.

Default: 30

Notes: This value must be shorter than the SLEE watchdog timeout period.
Example: createBucketExpiryDays = 30

downstreamOverloadSleepUSec

Syntax: downstreamOverloadSleepUSec = microsecs

Description: When a downstream process, beSync or beServer is overloaded, sleep for this period (in microseconds) before rechecking.

Type: Integer
Optionality: Optional (default used if not set).
Allowed:

Default: 100000

Notes: This value must be shorter than the SLEE watchdog timeout period.
Example: downstreamOverloadSleepUSec = 100000

handlers

Syntax: handlers = [
  "lib"
  ...
]

Description: Lists the beVWARS message handler plug-ins to load.

Type: Array
Optionality: Optional (default used if not set).
Allowed:
Chapter 3

Default:
Notes: This array must include handlers for messages from processes requesting billing actions.
For more information about handlers from other applications, see the associated technical guide.
Example: `handlers = [ "beVWARSCCDRHandler.so" ]`

**maxDownstreamQueueLength**

Syntax: `maxDownstreamQueueLength = num`
Description: The maximum number of pending events on beSync or beServer.
Type: Integer
Optionality: Optional (default used if not set).
Allowed: Default: 10000
Notes: When this number is exceeded, if events are queues on either of the processes, we sleep.
Example: `maxDownstreamQueueLength = 10000`

**maxOpenDialogTime**

Syntax: `maxOpenDialogTime = seconds`
Description: How long (in seconds) to try to open dialogs to the other SLEE processes.
Type: Integer
Optionality: Optional (default used if not set).
Allowed: Default: 5.0
Notes: Example: `maxOpenDialogTime = 5.0`

**maxSendReservationsToSync**

Syntax: `maxSendReservationsToSync = num`
Description: When beSync has requested all reservations, this is the number to send in one pass.
Type: Integer
Optionality: Optional (default used if not set).
Allowed: Default: 1000
Notes: Example: `maxSendReservationsToSync = 1000`

**maxTransactionsPerSet**

Syntax: `maxTransactionsPerSet = num`
Description: The number of Transactions and EDRs to try initially to fit into a TransactionSet written to the sync files.
Type: Integer
Optionality: Optional (default used if not set).
Allowed: 7
Default: 7
Notes: This TransactionSet will be passed across the SLEE on the other VWS, so must fit inside a SLEE event.
If this number of transactions does not fit the message, the message is re-encoded with fewer and fewer Transactions and EDRs per TransactionSet.
7 is used because 7.75 132 byte Transactions fit into a 1024 bytes SleeEvent.
Example: maxTransactionsPerSet = 7

minResyncReservationLength
Syntax: minResyncReservationLength = seconds
Description: Minimum reservation length (in seconds) before passing the reservation to the other Voucher and Wallet Server in a pair.
Type: Integer
Optionality: Optional (default used if not set).
Allowed: positive integer
Minimum 0
Default: 5
Notes: Set this parameter to reduce the amount of reservations which are sent where they will have expired by the time they are received by the other VWS.
Example: minResyncReservationLength = 5

plugins
Syntax: plugins = [ "lib" ][...]
Description: Lists the beVWARS event handler plug-ins to load.
Type: Parameter array
Optionality: Allowed: Default:
Notes: Where plug-ins are triggered by the same event, they will operate in the order they appear in this list.
For more information about plug-ins from other applications, see the associated technical guide.
Example: plugins = [ "beVWARSExpiry.so"
]

reservationExpiryCheckMilliseconds
Syntax: reservationExpiryCheckMilliseconds = millisecs
Description: The frequency (milliseconds) that the reservation expiry check occurs.
Type: Integer
Optionality: Optional (default used if not set).
Allowed:
Default: 10000
Notes:  
Example:  

reservationExpiryCheckMilliseconds = 5000

setLastActivationDateStates

Syntax:  

setDescription states = [states]

Description:  
A list of all old wallet states which will cause the 'Last Activation Date' for the wallet to be updated. The new wallet state in all these cases will be "ACTV" after the call has completed.

Type:  
String

Optionality:  
Optional (default used if not set).

Allowed:  
Valid values are: PREU, FROZ, DORM, SUSP, and TERM

Default:  
PREU

Notes:

Example:  

setLastActivationDateStates = [PREU,DORM]

useTimeFromClient

Syntax:  

useTimeFromClient = true|false

Description:  
If true, use the time from the incoming message (client date and usec).

Type:  
Boolean

Optionality:  
Optional (default used if not set).

Allowed:  

Default:  
true

Notes:  
This ensures that a given client message is treated identically on both VWSs, when it is re-sent. Otherwise, duplicate detection will cause the VWSs to get out of sync.

Example:  

useTimeFromClient = true

duplicateDetection parameters

Duplicate messages are possible, as the BeClient may switch to the auxiliary beServer after the original beServer has processed the message, but failed to return a response. The beVWARS detects duplicates by keeping a list of the client, clientMessageId and clientMsgTimestamps from messages received directly from the local beServer or received via beSync.

It is not necessary to store message identifiers permanently, as the BeClient switchover time is finite. For a given stream of messages, it is possible to tell that the client has not switched over.
Example: If the client is set to switch over at 1:00, but a message is received dated 3:00 from the beServer, we know that the BeClient has not switched over (or messages would not continue to be received via the beServer).

If we receive a message through the beServer dated 2:00, we know that we will not receive any other messages (from the BeClient) through the beServer dated 1:30 or earlier. Due to wallet locks, messages are not always received exactly in the beClientMessageTimestamp order.

directMaxDelaySeconds

Syntax: \texttt{directMaxDelaySeconds = seconds}

Description: The maximum delay (in seconds) before IDs received directly are removed from the main stream. IDs are kept for this time, in order to compare them with IDs from the converse stream.

Type: Integer

Optionality: Optional (default used if not set).

Allowed: 1.0

Notes: It is recommended that the delay be kept to a minimum, so that the timestamp order is not affected too greatly.

Example: \texttt{directMaxDelaySeconds = 1.0}

keepDirectSeconds

Syntax: \texttt{keepDirectSeconds = seconds}

Description: The time (in seconds) to keep IDs from messages received directly (through beServer), for comparison later with IDs from messages received through sync (through beSync).

Type: Integer

Optionality: Optional (default used if not set).

Allowed: 60.0

Notes: If a message has been kept much longer than the latest message received, we know that the BeClient has not switched beServers, so duplicates are not possible.

Example: \texttt{keepDirectSeconds = 60.0}

keepSyncSeconds

Syntax: \texttt{keepSyncSeconds = seconds}

Description: The time (in seconds) to leave the IDs in the duplicate map, to wait for a duplicate.

Type: Integer

Optionality: Optional (default used if not set).

Allowed: 60.0

Notes: Example: \texttt{keepSyncSeconds = 60.0}

syncMaxDelaySeconds

Syntax: \texttt{syncMaxDelaySeconds = seconds}

Description: The maximum delay (in seconds) before IDs are removed from the duplicate map. IDs are kept for this time, in order to compare them with IDs from the converse stream.
Type: Integer
Optionality: Optional (default used if not set).
Allowed: 1.0
Default: 1.0
Notes: It is recommended that the delay be kept to a minimum, so that the timestamp order is not affected too greatly.
Example: `syncMaxDelaySeconds = 1.0`

### groveller parameters

The groveller sub-section of the `beVWARS` provides the configuration for the groveller subsystem in `beVWARS` to query wallets and run plug-ins against them. Wallets are supplied by wallet id from `beGroveller`. For more information about how wallets are groveled, see *Background processing* (on page 6).

#### peerDatabaseLogin

**Syntax:**

```
peerDatabaseLogin = "login"
```

**Description:** If `peerDatabaseLogin` is not "", the groveller will perform a remote database query on the other VWS's database for information on the wallet. If inconsistent, another check is attempted after the number of seconds configured in `peerWalletCheckRetrySeconds`. This gives the remote VWS time to process and commit transactions. If the wallets are still inconsistent, a syslog message is produced.

**Type:**

- Integer

**Units:**

- Optional (default used if not set).

**Allowed:**

- ""

**Default:** ""

**Important:** This should not be used in production due to impact on performance.

**Example:** `peerDatabaseLogin = ""`

#### peerWalletCheckRetrySeconds

**Syntax:**

```
peerWalletCheckRetrySeconds = seconds
```

**Description:** How long (in seconds) to wait after the peer VWS's information on a wallet, if found to be inconsistent. If the wallet is still inconsistent after this period a syslog message is produced.

**Type:**

- Integer

**Optionality:** Optional (default used if not set).

**Allowed:**

- 60

**Default:** 60

**Notes:** This period should be long enough for locally generated operation message to be sent to the remote VWS, and committed to the database.

**Example:** `peerWalletCheckRetrySeconds = 60`

#### periodMsec

**Syntax:**

```
periodMsec = millisecs
```

**Description:** The minimum number of milliseconds between groveling wallets.

**Type:** Integer
Optionality: Optional (default used if not set).
Allowed: 0  Send a new request as soon as possible.
        positive integer
Default: 1000
Notes: This parameter is restricted by requestHighWaterMark (on page 94).
      Setting request frequency to 0 will impact the normal VWS processing speed.
      This parameter will have no effect if groveling is disabled (for example, if enableGrovelling (on page 35)) is set to false).
Example: periodMsec = 1200

requestHighWaterMark
Syntax: requestHighWaterMark = num
Description: The maximum number of outstanding requests queued on the SLEE waiting on this beVWARS instance before no requests are sent to beGroveller for wallets to grovel.
Type: Integer
Optionality: Optional (default used if not set).
Allowed:
Default: 1
Notes: If the number of outstanding requests in the SLEE queue is less than or equal to half of requestHighWaterMark, then this beVWARS instance will try to grovel a wallet between every request it processes.
      If the number of outstanding requests in the SLEE queue is more than half of requestHighWaterMark, wallets will be groveled with decreasing frequency proportional to the queue length, until only one wallet is groveled every for every 10 requests from beServer. The proportion is calculated to approximately increase the total number of requests (SLEE queue + grovel requests) to requestHighWaterMark.
Examples: These examples assume requestHighWaterMark = 100.
  • If queue length = 25, beVWARS will process one grovel request for each SLEE queue request (that is, 50 of every 100 requests will be grovel requests).
  • If queue length = 50, beVWARS will process one grovel request for each SLEE queue request (that is, 50 of every 100 requests will be grovel requests).
  • If queue length = 80, beVWARS will process two grovel requests for every eight SLEE queue requests (that is, 20 of every 100 requests will be grovel requests).
  • If queue length = 90, beVWARS will process one grovel request for every 10 SLEE queue requests (that is, 10 of every 100 requests will be grovel requests).
  • If queue length = 100, beVWARS will not process any grovel requests.
Example: requestHighWaterMark = 100

requestTimeout
Syntax: requestTimeout = seconds
Description: The maximum number of seconds to wait for a response after sending a request to beGroveller for another batch of wallet IDs to grovel, before timing out the request.
Chapter 3

Type: Integer
Optionality: Optional (default used if not set).
Allowed: positive integer

-1 Do not time out requests.

Default: 30

Notes: If a request is timed out, beVWARS will log a Warning level error to syslog and will resend the request.

If errors are being logged to the syslog indicating timeouts, try:
- Setting `walletLowWaterMark` (on page 95) to a higher value
- Setting `requestTimeout` to a higher value
- If there are not as many beGroveler processes as beVWARS processes, adding the number of beGroveler processes (this can be done until there are as many beGroveler processes as beVWARS processes)
- Setting `maxIDsPerResponse` (on page 55) to a higher value

Example: `requestTimeout = 30`

`secondaryConnectionDelaySeconds`

Syntax: `secondaryConnectionDelaySeconds = seconds`

Description: The number of seconds to wait before enabling the beGroveler to process wallets on the secondary VWS if the primary VWS cannot be contacted. Ensures that wallet processing by the beGroveler is not started on the secondary VWS if the primary VWS is down for a short period of time only.

Type: Integer
Optionality: Optional (default used if not set).

Default: 900

Notes: The configured delay is applied when the secondary VWS beGroveler is started and when the client connection from the secondary VWS beGroveler to the primary beServer is lost.

Example: `secondaryConnectionDelaySeconds = 900`

`walletLowWaterMark`

Syntax: `walletLowWaterMark = num`

Description: The number of outstanding wallet IDs to grovel, before sending a request to beGroveler for another batch of wallet IDs to grovel.

Type: Integer
Optionality: Optional (default used if not set).

Default: 100

Notes: Example: `walletLowWaterMark = 100`

`voucherCache parameters`

The `voucherCache` sub-section of the `beVWARS` parameters defines the voucher cache.
checkBeforeFlush

Syntax: checkBeforeFlush = true | false
Description: If true, vouchers are checked against the database before flushing. If they are different, an error is sent to syslog.
Type: Boolean
Units: Optionality: Optional (default used if not set).
Allowed: true, false
Default: false
Important: This should not be used in production due to impact on performance.
Example: checkBeforeFlush = false

maxLoopSize

Syntax: maxLoopSize = num
Description: The number of vouchers in the voucher cache for beVWARS to process at one time before pausing. This enables beVWARS to pause and respond to other requests, instead of attempting to process the whole cache at once.
Type: Integer
Optionality: Optional (default used if not set).
Allowed: Positive integers Number of vouchers to process in a batch.
-1 Process whole cache at once.
Default: 10000
Notes: This parameter should be set if maxSize and maxAgeSeconds are set to -1 (which is likely to cause a large cache).
If this number is set too high (or to -1) the SLEE watchdog may restart beVWARS unnecessarily. Too low, and beVWARS will waste CPU polling the SLEE unnecessarily.
Example: maxLoopSize = 5000

maxSize

Syntax: maxSize = MB
Description: The maximum size of the beVWARS voucher cache.
Type: Integer
Units: MB
Optionality: Optional (default used if not set).
Allowed: 0 or a positive integer.
Default: 10000
Notes: Example: maxSize = 10000

walletCache parameters

The walletCache sub-section of the beVWARS parameters defines the wallet cache.

checkBeforeFlush

Syntax: checkBeforeFlush = true | false
Description: If true, wallets are checked against the database before flushing. If they are different, an error is sent to syslog.
Chapter 3

syncWriter parameters

The syncWriter sub-section of the beVWARS parameters defines how the beVWARS syncWriter writes sync files.

maxRecordsPerFile

Syntax:  
maxRecordsPerFile = num

Description:  
The maximum number of records in a sync file.

Type:  
Integer

Optionality:  
Optional (default used if not set).
**maxSecondsPerFile**

**Syntax:** \( \text{maxSecondsPerFile} = \text{seconds} \)

**Description:** The maximum number of seconds to hold a sync file open.

**Type:** Integer

**Optionality:** Optional (default used if not set).

**Allowed:**

- **Default:** 2

**Notes:**

**Example:** \( \text{maxSecondsPerFile} = 4 \)

---

**dbWriter parameters**

The **dbWriter** sub-section of the **beVWARS** parameters defines how the beVWARS dbWriter buffers and writes:

- Data updates to the E2BE database
- EDRs to the filesystem

**Note:** All buffers and the EDR cache are flushed whenever one of the following conditions occurs:

- One of the buffers is full
- The EDR cache is full
- The current flush period has ended
- The beVWARS writer subsystem is told to flush and commit (on shutdown for example)

**balanceCreateBufferSize**

**Syntax:** \( \text{balanceCreateBufferSize} = \text{num} \)

**Description:** The number of items in a buffer before beVWARS will flush it.

**Type:** Integer

**Optionality:** Optional (default used if not set).

**Allowed:**

- **Default:** 1000

**Notes:**

**Example:** \( \text{balanceCreateBufferSize} = 1500 \)

**balanceDeleteBufferSize**

**Syntax:** \( \text{balanceDeleteBufferSize} = \text{num} \)

**Description:** The number of deletes in a buffer before beVWARS will flush it.

**Type:** Integer

**Optionality:** Optional (default used if not set).

**Allowed:**

- **Default:** 1000

**Notes:**
Example: \texttt{balanceDeleteBufferSize} = 1000

\textbf{balanceUpdateBufferSize}  

**Syntax:** \texttt{balanceUpdateBufferSize} = \textit{num}  
**Description:** The number of updates in a buffer before beVWARS will flush it.  
**Type:** Integer  
**Optionality:** Optional (default used if not set).  
**Allowed:**  
**Default:** 1000  
**Notes:**  
**Example:** \texttt{balanceUpdateBufferSize} = 1000

\textbf{bucketCreateBufferSize}  

**Syntax:** \texttt{bucketCreateBufferSize} = \textit{num}  
**Description:** The number of bucket creates in a buffer before beVWARS will flush it.  
**Type:** Integer  
**Optionality:** Optional (default used if not set).  
**Allowed:**  
**Default:** 1000  
**Notes:**  
**Example:** \texttt{bucketCreateBufferSize} = 1000

\textbf{bucketDeleteBufferSize}  

**Syntax:** \texttt{bucketDeleteBufferSize} = \textit{num}  
**Description:** The number of bucket deletes in a buffer before beVWARS will flush it.  
**Type:** Integer  
**Optionality:** Optional (default used if not set).  
**Allowed:**  
**Default:** 1000  
**Notes:**  
**Example:** \texttt{bucketDeleteBufferSize} = 1000

\textbf{bucketUpdateBufferSize}  

**Syntax:** \texttt{bucketUpdateBufferSize} = \textit{num}  
**Description:** The number of bucket updates in a buffer before beVWARS will flush it.  
**Type:** Integer  
**Optionality:** Optional (default used if not set).  
**Allowed:**  
**Default:** 1000  
**Notes:**  
**Example:** \texttt{bucketUpdateBufferSize} = 1000

\textbf{cdrOutputDirectory}  

**Syntax:** \texttt{cdrOutputDirectory} = \textasciitilde \textit{dir}  
**Description:** Directory where EDRs are written to.  

---

Chapter 3, Background Processes 99
Type: String
Optionality: Optional (default used if not set).
Allowed: 
Default: "/IN/service_packages/E2BE/logs/CDR"
Notes: EDRs will be stored in this directory until they are moved by another process.
Example: cdrOutputDirectory = "/var/EDRs/UBE/"

flushPeriod
Syntax: flushPeriod = seconds
Description: The maximum number of seconds between flushes.
Type: Integer
Optionality: Optional (default used if not set).
Allowed: 
Default: 10
Notes: 
Example: flushPeriod = 20

voucherUpdateBufferSize
Syntax: voucherUpdateBufferSize = num
Description: The number of voucher updates in a buffer before beVWARS will flush it.
Type: Integer
Optionality: Optional (default used if not set).
Allowed: 
Default: 1000
Notes: 
Example: voucherUpdateBufferSize = 1000

voucherCreateBufferSize
Syntax: voucherCreateBufferSize = num
Description: The number of voucher creates in a buffer before beVWARS will flush it.
Type: Integer
Optionality: Optional (default used if not set).
Allowed: 
Default: 1000
Notes: 
Example: voucherCreateBufferSize = 1000

voucherDeleteBufferSize
Syntax: voucherDeleteBufferSize = num
Description: The number of voucher deletes in a buffer before beVWARS will flush it.
Type: Integer
Optionality: Optional (default used if not set).
Allowed: 
Default: 1000
Notes: 

Example: voucherDeleteBufferSize = 1000

walletCreateBufferSize
Syntax: walletCreateBufferSize = num
Description: The number of wallet creates in a buffer before beVWARS will flush it.
Type: Integer
Optionality: Optional (default used if not set).
Allowed: 
Default: 1000
Notes: 
Example: walletCreateBufferSize = 1000

walletDeleteBufferSize
Syntax: walletDeleteBufferSize = num
Description: The number of wallet deletes in a buffer before beVWARS will flush it.
Type: Integer
Optionality: Optional (default used if not set).
Allowed: 
Default: 1000
Notes: 
Example: walletDeleteBufferSize = 1000

walletUpdateBufferSize
Syntax: walletUpdateBufferSize = num
Description: The number of wallet updates in a buffer before beVWARS will flush it.
Type: Integer
Optionality: Optional (default used if not set).
Allowed: 
Default: 1000
Notes: 
Example: walletUpdateBufferSize = 1000

Tracing parameters

The tracing parameters allow tracing to be performed for individual wallet IDs on selected be clients.
Where more than one criteria (wallet and client) is configured for tracing then the message must satisfy all criteria (logical AND) for tracing/debug to activate.

beClients
Syntax: beClients = ["client1", "client2", ...]
Description: List of BE client names to trace.
Type: Array, String
Optionality: Optional if walletIDs parameter supplied, mandatory if walletIDs not supplied.
Allowed: Any beClient.
Default: None
Notes: The names are converted to a unique BE client hash ID - which is the same mechanism employed by the beVWARS for referencing BE clients.

Example:
```
beClients = [
    "ccsBeOrb",
    "PIbeClient"
]
```

defaultLevel

Syntax: `debugLevel = "level"`
Description: The debug level/filter, is equivalent to DEBUG environment variable.
Type: String
Optionality: Optional (default used if not set).
Allowed: Any of the DEBUG options.
Default: "all"
Notes: This is a comma separated string. See traceDebugLevel in ACS Technical Guide for more information.
Example: `debugLevel = "all"

enabled

Syntax: `enabled = true|false`
Description: The tracing activation switch to allow tracing of selected wallet and/or be client activity.
Type: Boolean
Optionality: Optional (default used if not set).
Allowed: true, false
Default: false
Notes: Example: `enabled = true`

walletIds

Syntax: `walletIds = [ID1, ID2, ...]`
Description: List of subscriber wallet ids we want to trace.
Type: Array, Integer
Optionality: Optional if beClient parameter supplied, mandatory if beClient not supplied.
Allowed: Any valid wallet ID.
Default: None
Notes: To obtain the wallet id(s) for a given CLI/subscriber use the showCLI.sh script on the BE where tracing is to occur.
Example: `walletIds = [382, 385]

Example configuration

This is an example beVWARS section `eserv.config` on a VWS node (comments have been removed).
```
beVWARS = {
```
voucherReservationPeriodSeconds = 120
useTimeFromClient = true
maxTransactionsPerSet = 7
maxOpenDialogTime = 5.0
maxDownstreamQueueLength = 10000
downstreamOverloadSleepUSec = 100000
minResyncReservationLength = 5
createBucketExpiryDays = 30
maxSendReservationsToSync = 1000
reservationExpiryCheckMilliseconds = 10000

walletCache = {
  maxSize = 10000
  checkBeforeFlush = false
  maxLoopSize = 500
}

voucherCache = {
  maxSize = -100000
  checkBeforeFlush = false
  maxLoopSize = 500
}

groveller = {
  periodMsec = 1200
  requestHighWaterMark = 1
  walletLowWaterMark = 100
  requestTimeout = 300
  peerDatabaseLogin = ""
  peerWalletCheckRetrySeconds = 60
  secondaryConnectionDelaySeconds = 900
}

duplicateDetection = {
  keepDirectSeconds = 60.0
  keepSyncSeconds = 60.0
  directMaxDelaySeconds = 1.0
  syncMaxDelaySeconds = 1.0
}

setLastActivationDateStates = [
  [PREU]
]

plugins = [
  "beVWARSExpiry.so"
]

handlers = [
  "beVWARSCCDRHandler.so"
]

syncWriter = {
  maxRecordsPerFile = 100
  maxSecondsPerFile = 2
}

dbWriter = {
  flushPeriod = 10
  cdrOutputDirectory = "/IN/service_packages/E2BE/logs/CDR"
  balanceCreateBufferSize = 1000
balanceUpdateBufferSize = 1000
balanceDeleteBufferSize = 1000
bucketCreateBufferSize = 1000
bucketUpdateBufferSize = 1000
bucketDeleteBufferSize = 1000
walletCreateBufferSize = 1000
walletUpdateBufferSize = 1000
walletDeleteBufferSize = 1000
voucherCreateBufferSize = 1000
voucherUpdateBufferSize = 1000
voucherDeleteBufferSize = 1000
}
tracing = {
    enabled = true
    debugLevel = "all"
    walletIds = [
        382,
        385
    ]
    beClients = [
        "ccsBeOrb",
        "PIbeClient"
    ]
})
setLastActivationDateStates = [
    "PREU"
]
} # BE.beVWARS

Output

Each beVWARS writes error messages to the system messages file, and also writes additional output to its own log file. By default this is:

/IN/service_packages/E2BE/tmp/beVWARS0.log

Note: The actual name will be different for each beVWARS process.

beVWARSCCDRHandler

Purpose

The beVWARSCCDRHandler provides a specific EDR-generating function. This is generally used where no other process in an interaction will produce an EDR, but an EDR should still be generated.

Example: If a voucher redeem fails because the voucher cannot be found on any Voucher and Wallet Server, the client process will send a request to the beVWARSCCDRHandler to write an EDR for the failed voucher redeem.

Startup

beVWARSCCDRHandler.so is included in the beVWARS by specifying it in the handlers array.

For more information about the handlers array, see handlers (on page 88).

Configuration

This binary has no specific configuration.
beVWARSEpiry

Purpose

beVWARSEpiry monitors subscriber accounts and wallets, checking for subscriber accounts and wallets which have passed their expiry date. If it finds a subscriber account or wallet which requires expiring, it processes the record as configured.

Startup

If beVWARSEpiry is configured in eserv.config, it is started by beVWARS when beVWARS is initialized. It is included in the beVWARS handlers section.

```python
handlers = [  
    "beVWARSEpiry.so"
]
```

For more information about beVWARS:

- Plug-ins, see Plug-ins (on page 86).
- Handlers section, see handlers (on page 88)

**Note:** Other handlers may also be included in the handlers list.

Configuration

beVWARSEpiry accepts the following parameters from eserv.config.

```ruby
beVWARSEpiry = {  
    expireNegativeBuckets = true|false  
    removeEmptyBuckets = true|false  
    expireBucketsForExpiredWallets = true|false  
    expireAtMidnightTZ = "timezone"
}
```

Parameters

Here are the available parameters in the beVWARSEpiry section of the eserv.config.

expireAtMidnightTZ

**Syntax:**

```plaintext
expireAtMidnightTZ = "timezone"
```

**Description:** Sets wallets and buckets to expire at midnight for the time zone specified.

**Type:** String

**Optionality:** Optional (default used if not set).

**Allowed:** The time zone part of the parameter must be typed in a form that the operating system recognizes.

Alternatively you can select a time zone from the operating system's list. To view top-level time zone names, type `ls /usr/share/lib/zoneinfo` from a shell.

To see second-level time zone names type `ls /usr/share/lib/zoneinfo TopLevelName/`. For example, to verify that the operating system recognizes a time zone name for DeNoranha, in Brazil, you would type `ls /usr/share/lib/zoneinfo/Brazil/`. DeNoranha is listed, so the time zone name would be "Brazil/DeNoranha".

**Default:** false (do not modify expiry calculation).
Notes: A list of time zones can be found in the Time Zones appendix of ACS Technical Guide.

This parameter does not affect the expiry calculations of periodic charge buckets.

Example: An account is created at 2 p.m. on 5 September 2014 and is set to have a life span of 24 days.
If expireAtMidnightTZ = "Asia/Vladivostok" is included, the account will expire on 29 September 2014 at midnight, Vladivostok time.
If this parameter is omitted, the account will expire on 29 September 2014 at 2 p.m.

expireBucketsForExpiredWallets
Syntax: expireBucketsForExpiredWallets = true|false
Description: Controls whether wallet expiry triggers bucket expiry.
If true, any buckets under the wallet will be expired when the wallet expires.
Type: Boolean
Optionality: Optional (default used if not set).
Allowed: true All buckets with a positive or zero value in a wallet will be expired when the wallet is expired, even if the buckets are not due to expire yet.
false Buckets are expired when their own expiry date passes.

Note: This means the wallet will not be deleted from the system, but will instead be set to Removed state and kept until the last bucket is expired.

Default: false

Notes: If expireNegativeBuckets (on page 106) is set to false, buckets with a negative value will not be deleted, regardless of the value of this parameter.
Using this parameter will remove any positive value the wallet holds when the wallet expires.
Example: expireBucketsForExpiredWallets = false

expireNegativeBuckets
Syntax: expireNegativeBuckets = true|false
Description: Whether or not to expire buckets which have a negative value.
Type: Boolean
Optionality: Optional (default used if not set).
Allowed: true Buckets with negative values are expired when their expiry date passes.

Buckets with negative values (where the subscriber is in debit), are expired when their expiry date passes.
false Buckets are expired when their expiry date passes and they have a positive or 0 balance.

Note: This means wallets with negative balances will not be deleted from the system, but will instead be set to Removed state and kept until the last bucket is expired.
Default: false
Notes: This parameter is designed to enable the Telco to keep the wallet until all outstanding money has been recovered from the subscriber.
Example: expireNegativeBuckets = false

removeEmptyBuckets
Syntax: removeEmptyBuckets = true|false
Description: Whether or not to remove buckets when they have a value of 0.
Type: Boolean
Optionality: Optional (default used if not set).
Allowed: false Leave buckets to expire as normal.
true If true, any buckets with 0 value will be removed.
Default: false
Notes: Setting this to false does not stop beVWARSExpiry removing buckets for expired wallets if expireBucketsForExpiredWallets (on page 106) is set to true.
Example: removeEmptyBuckets = false

Example configuration
This is an example of the beVWARSExpiry section of an eserv.config file from a VWS (comments have been removed).

```
beVWARSExpiry = {
    expireNegativeBuckets = false
    removeEmptyBuckets = false
    expireBucketsForExpiredWallets = false
    expireAtMidnightTZ = "Asia/Vladivostok"
}
```

Failure
If beVWARSExpiry fails, it will not trigger expiry events for any Expiry plug-in. When beVWARSExpiry recovers, it will process as normal, and will catch up with any expired wallets or buckets.

Output
The beVWARSExpiry writes error messages to the system messages file, and also writes additional output to the following default (may vary as per configuration):

```
/IN/service_packages/E2BE/tmp/beVWARSExpiry.log
```

beVWARSMergeBuckets

Purpose
beVWARSMergeBuckets is a plug-in library for beVWARS.

This beVWARS plug-in merges buckets in the same balance when there are too many buckets in the wallet. If there are too many buckets the message detailing the wallet contents will not fit in a 1024 byte SLEE event and may cause errors.
Merging begins with the balances that have the most buckets. In each balance, the bucket with the earliest expiry has its value added to the next bucket, then it is removed. This is repeated until the wallet has the maximum allowed number of buckets left.

**Configuration**

beVWARSMergeBuckets accepts the following parameters from `eserv.config`.

```
maxBuckets = num
triggerPlugins = true|false
```

**Parameters**

Here are the available parameters in the `beVWARSMergeBuckets` section of the `eserv.config`.

**maxBuckets**

- **Syntax:** `maxBuckets = num`
- **Description:** The maximum number of buckets a wallet may have.
- **Type:** Integer
- **Units:**
- **Optionality:** Optional (default used if not set).
- **Allowed:**
  - `-1` No maximum.
  - `positive integer` Maximum number of buckets.
- **Default:** `-1`
- **Notes:**
- **Example:** `maxBuckets = -1`

**triggerPlugins**

- **Syntax:** `triggerPlugins = true|false`
- **Description:** When we merge buckets (update the value of one and delete the other), should we trigger other beVWARS plug-in.
- **Type:** Boolean
- **Units:**
- **Optionality:** Optional (default used if not set).
- **Allowed:**
- **Default:** `false`
- **Notes:**
- **Example:** `triggerPlugins = false`

**Example configuration**

This is an example of the `beVWARSExpiry` section of an `eserv.config` file on a VWS (comments have been removed).

```
beVWARSMergeBuckets = {
    maxBuckets = -1
    triggerPlugins = false
}
```
**cmnPushFiles**

**Purpose**

cmnPushFiles transfers files to specific directories on the SMS from SLCs and VWSs. The files transferred include:

- EDRs
- PIN logs

**Note:** Other Oracle applications also use their own instances of this process.

**Startup**

This task is started by entry scp1 in the initab, using the shell script:

```
/IN/service_packages/SMS/bin/cmnPushFilesStartup.sh
```

**Configuration**

cmnPushFiles accepts the following command-line options:

**Usage:**

```
cmnPushFiles -d dir [-o dir [-a age]] [-f dir] [-F] [-P prefix] [-S suffix] -h host 
bitrate] [-T] [-x] [-e] [-w seconds]
```

The available parameters are:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-d</td>
<td></td>
<td>Destination directory for files on remote machine. Example: The directory on SLC where the cmnPushFiles looks for the files to be sent to the SMS.</td>
</tr>
<tr>
<td>-o</td>
<td>File deleted</td>
<td>Transferred directory.</td>
</tr>
<tr>
<td>-a</td>
<td>Never delete files</td>
<td>Age of transferred files before being deleted. This parameter only relevant when -o option is specified.</td>
</tr>
<tr>
<td>-f</td>
<td>none</td>
<td>Retry directory.</td>
</tr>
<tr>
<td>-F</td>
<td>Do not use</td>
<td>Use fuser to not move files in use.</td>
</tr>
<tr>
<td>-P</td>
<td>none</td>
<td>File prefix.</td>
</tr>
<tr>
<td>-S</td>
<td>none</td>
<td>File suffix.</td>
</tr>
<tr>
<td>-h</td>
<td>none</td>
<td>Remote hostname.</td>
</tr>
<tr>
<td>-r</td>
<td>none</td>
<td>Remote directory prefix. Note: Required if -d is relative directory.</td>
</tr>
<tr>
<td>-p</td>
<td>2027</td>
<td>Port on remote machine on which the cmnReceiveFiles will listen for receiving files. Note: -1 for stdin/stdout.</td>
</tr>
<tr>
<td>-s</td>
<td>15</td>
<td>Sleep period in seconds.</td>
</tr>
<tr>
<td>-R</td>
<td>15</td>
<td>Seconds before Initial retry period in seconds.</td>
</tr>
<tr>
<td>-M</td>
<td>900</td>
<td>Maximum retry period in seconds.</td>
</tr>
<tr>
<td>-C</td>
<td>1800</td>
<td>Cleanup period in seconds.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Default</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>---------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>-t</td>
<td>none (no</td>
<td>Throttles transfer to ( nnn ) bits per second.</td>
</tr>
<tr>
<td></td>
<td>throttling)</td>
<td></td>
</tr>
<tr>
<td>-T</td>
<td>off (non-</td>
<td>Tree move: recursive into subdirectories.</td>
</tr>
<tr>
<td></td>
<td>recursive)</td>
<td></td>
</tr>
<tr>
<td>-x</td>
<td>On (use</td>
<td>Do not use hostname-prefixing on remote filenames.</td>
</tr>
<tr>
<td></td>
<td>prefixing)</td>
<td></td>
</tr>
<tr>
<td>-e</td>
<td>Daemon mode</td>
<td>Non-daemon mode. Execute file transfer only once, then exit.</td>
</tr>
<tr>
<td>-w</td>
<td>30</td>
<td>Time to wait for success in seconds.</td>
</tr>
</tbody>
</table>

**Example:**

```bash
cmnPushFiles -d /IN/service_packages/SMS/cdr/closed -f /IN/service_packages/SMS/cdr/retry -r /IN/service_packages/SMS/cdr/received -h prodsmp1.telcoexample.com -s 10 -p 2028 -S cdr -w 20
```

**Parameters**

Here are the available parameters in the `cmnPushFiles` section of the `eserv.config` file.

**CDR**

**Syntax:**

```
CDR = [
    "param", "value"
    [...]  
]
```

**Description:** Arguments to `cmnPushFiles` when used to send EDRs to SMS.

**Type:** Array

**Optionality:** Optional (default used if not set).

**Allowed:** See `cmnPushFiles` documentation in *SMS Technical Guide*.

**Default:**

**Notes:**

**Example:**

```
CDR = [
    "-d", "/IN/service_packages/E2BE/logs/CDR-out"
    "-r", "/IN/service_packages/CCS/logs/CDR-in"
    "-h", "smplprod"
    "-f"
]
```

**-d**

**Syntax:**

```
"-d", "dir"
```

**Description:** Local source directory.

**Type:** String

**Optionality:**

**Allowed:**

**Default:**

**Notes:**

**Example:**

```
"-d", "/IN/service_packages/E2BE/logs/CDR-out"
```
-r
Syntax: "-r", "dir"
Description: Remote destination directory.
Type: String
Optionality: 
Allowed: 
Default: 
Notes: 
Example: "-r", "/IN/service_packages/CCS/logs/CDR-in"

-h
Syntax: "-h", "host"
Description: Full host name and domain of the SMS machine.
Type: String
Optionality: 
Allowed: 
Default: 
Notes: 
Example: "-h", "smplprod"

-F
Syntax: -F
Description: Do not send the file if a process is currently using it.
Type: Boolean
Optionality: Optional (file sent if not set).
Allowed: 
Default: 
Notes: 
Example: -F

Example configuration
This is an example of the cmnPushFiles section of an eserv.config file on a VWS (comments have been removed).

```plaintext
cmnPushFiles = {
    CDR = [{
        "-d", "/IN/service_packages/E2BE/logs/CDR-out"
        "-r", "/IN/service_packages/CCS/logs/CDR-in"
        "-h", "smplhostname"
        "-F"
    }
}
```

Failure
If cmnPushFiles fails, EDRs will accumulate in:
/IN/service_packages/SMS/cdr/current/
cmnPushFiles will send error messages to the syslog and the cmnPushFiles log.

**Output**

The cmnPushFiles writes error messages to the system messages file, and also writes additional output to this default location:

/IN/service_packages/SMS/tmp/cmnPushFiles.log

**Event Storage Interface**

**Overview**

The event storage interface stores events to be sent to a different SLEE interface at a future time. When the event reaches the time for it to be sent, the stored event is sent to the specified SLEE interface, and waits for a response, which can be one of the following:

- Any event other than a DIALOG_CLOSED, the event is considered to be delivered, and remove it from the queue.
- A DIALOG_CLOSED, some kind of problem occurred and the event will be retried later.

To prevent the storage interface from spamming the outbound interface with events, the storage interface accepts a throttle message, which will put a gap between events to the interface specified in the message.

**Error and throttle flow**

Here is an example flow showing the interaction between the plug-in and the beServiceTrigger when an event that is to be sent immediately, encounters multiple failures: first the SLC is down, and then there is a second failure is due to throttling.
Send later flow

Here is an example flow showing the interaction between the plug-in and the beServiceTrigger when there an event that is to be sent later.

```
beVWARS  beServiceTrigger  TimeNotificationI
        |                  |
        v                  v
    SendEventAt (DapSendRequest, later)  SendEventAtAck
    \
    Time passes
    \
    DapSendRequest
    \
    Message is sent
    \
    DapSendEventResponse
    \
    Message is complete
```

Crash flow

Here is an example flow showing the interaction between the plug-in and the beServiceTrigger when there an event that is to be sent immediately, encounters a beServiceTrigger failure.

```
beVWARS  beServiceTrigger  TimeNotificationI
        |                  |
        v                  v
    SendEventAt (DapSendRequest, later)  SendEventAtAck
    \
    Time passes
    \
    DapSendRequest
    \
    Message is sent
    \
    DialogClosed
    \
    Re-insert the event into the queue for retry later
```

Event Storage SLEE Events

Trigger events for event storage, plus parameters are:

- `SenSendEventAt`:
  - `When`
  - `InterfaceName`
  - `EventToSend`
- `SendEventAck`
  - `Success`
- `ThrottleSending`
  - `TimeBetweenEvents`
libbeMsgRouterDefault

Purpose

libbeMsgRouterDefault is a beServer plug-in which determines which beVWARS to direct ESCHER messages to.

Startup

libclientBcast is used by beServer process if its configuration includes the library. To be used, it must be included in the `messageRoutingPlugins` array as shown:

```c
messageRoutingPlugins = [  
    "libbeMsgRouterDefault.so"
]
```

For more information about the beServer `messageRoutingPlugins` section, see `messageRoutingPlugins` (on page 63).

Configuration

The `libbeMsgRouterDefault`'s configuration is read from the `beServer` section of `eserv.config`. `libbeMsgRouterDefault` supports the following parameters:

- All parameters in `msgRouterDefault` (on page 63)
- `routingVoucherNumberLength` (on page 67)

libBeClientIF

Purpose

The `libBeClientIF` provides an interface to one or more VWS Voucher and Wallet Servers.

Startup

The `libBeClientIF` is a runtime dependency of processes which need to talk to the VWS Voucher and Wallet Servers.

Configuration

The `libBeClientIF`'s configuration is usually read from the section which configures the process which is using it. `libBeClientIF` supports the following parameters.

- `beLocationPlugin` (on page 35)
- `clientName` (on page 48)
- `heartbeatPeriod` (on page 48)
- `messageTimeoutSeconds` (on page 49)
- `maxOutstandingMessages` (on page 49)
- `reportPeriodSeconds` (on page 51)
- `connectionRetryTime` (on page 48)
- `plugins` (on page 50)
- `notEndActions` (on page 49)
Notes:
- Any process which is using the libBeClientIF can use these parameters.
- beLocationPlugin is located in the shared parameters section.

**libclientBcast**

**Purpose**

libclientBcast is used by BeClient processes to send messages which may be answered by any Voucher and Wallet Server. The request is sent to one of the VWSs in all Voucher and Wallet Server pairs at once. The plug-in is activated by sending a message to BE ID 0.

This process is specifically used to redeem vouchers, and vouchers cannot be redeemed if this library is not loaded.

**Startup**

libclientBcast is used by BeClient processes if their configuration includes the library.

If libclientBcast is used by BeClient and ccsBeOrb, it is included in the plugins array as shown:

```json
plugins = [
    {
        config="",
        library="libclientBcast.so",
        function="makeBroadcastPlugin"
    }
]
```

For more information about the BeClient plugins section, see plugins (on page 50).

**Configuration**

The libBeClientIF's configuration is usually read from the section which configures the process which is using it. libBeClientIF supports the aggregateNAckCodes (on page 47) parameter.

**libbeEventFactory**

**Purpose**

libbeEventFactory is the common library used to create SLEE events from ESCHER messages. It is required by the system and should not be removed.

**Startup**

libbeEventFactory is used by a number of processes on the VWS. No startup configuration is required for this library to be used.

**Configuration**

This binary has no specific configuration.
Tools and Utilities Overview

Introduction

This chapter provides a description of the operational programs or executables which can used on the VWS. All of these processes are performed when needed.

In this chapter

This chapter contains the following topics.
VWS Correction Tool

VWS Correction Tool

Purpose of tool

This tool can be used to restore consistent data in the system following a software fault or configuration error, without causing further outage or lost service for any node in the system.

This should be used for making small numbers of changes to fields that cannot be maintained via the User Interface or Provisioning Interface.

Rollback

There is no rollback as such. The commands can be run again with pre-corrected data to reach the previous state.

About minimizing tool impact

Any changes to the VWS database will affect the performance of the server. To mitigate performance degradation, several of the configuration parameters can be used to effectively throttle the tool.

The operator can:

- Limit number of concurrent commands executing in one binary by using the maxQueueSize configuration parameter. The tool will not issue new requests unless there are currently less than or equal to maxQueueSize commands in flight.
- Impose time delay between successive commands by using the maxCommandsPerSecond configuration parameter. The tool will issue this maximum number of commands per second. When used in conjunction with the maxQueueSize parameter, allows an even tighter throttle if required. For example, with a maxQueueSize of 1, the tool can still direct a strong volume commands, but if it is further moderated by a setting such as maxCommandsPerSecond=(say) 2 or even 1, then the traffic can be brought under tighter control.
- Queue or reject concurrent requests to same business object by using the queueUpdatesToSameObject configuration parameter. The tool can either allow concurrent or serial commands to run against the same business object. The default is false, which means that a file of commands for the same wallet will be executed serially, true, the commands are executed in parallel.
Starting the commands

For individual changes, the commands can be typed directly into the command line.

For many changes, the command lines can be entered into a batch file which is then executed from the command line.

eserv.config parameters

clientName

Syntax: clientName = "value"
Description: The unique client name to connect to the database with.
Type: String
Optionality: Optional (default used if not set).
Allowed: "ccsAccount"
Default: "ccsAccount"
Notes: Only one connections with the same name is allowed.
Example: clientName = "nccdemo-dev-ccsVWSCorrection"

heartbeatPeriod

Syntax: heartbeatPeriod = value
Description: The number of microseconds since previous message before fail over to the other VWS.
Type: Integer
Optionality: Optional (default used if not set).
Allowed: 0 for no heartbeating.
Default: 30000000 (30 seconds)
Notes: If no heartbeat or other messages received in this period we switch to the other VWS in the pair on the assumption that the current VWS has failed.
Example: heartbeatPeriod = 10000000

connectionRetryTime

Syntax: connectionRetryTime = value
Description: The number of seconds before we try to reconnect.
Type: Integer
Optionality: Optional (default used if not set).
Allowed: Values
Default: 5
Notes: Values
Example: connectionRetryTime = 2

plugins

Syntax: plugins = value
Description: Identifies which plugins to load.
Type: Array
Optionality: Optional (default used if not set).
Allowed:
Default: [] (empty, no plugins)
Notes: Not currently used, for future potential use.
Example: plugins = []

Syntax: billingEngines = value
Description: beLocationPlugin values override.
Type: Array
Optionality: Optional (default used if not set).
Allowed: beLocationPlugin billing engine values
Notes: Used to override the beLocationPlugin that would normally load the connection details from the DB.
Example:
billingEngines = [
    {id = 1, primary = { ip="PRIMARY_BE_IP", port=1500 }, secondary =
        { ip="SECONDARY_BE_IP", port=1500 }
    }
]

maxQueueSize

Syntax: maxQueueSize = value
Description: Number of concurrent commands executing in one binary.
Type: Integer
Optionality: Mandatory.
Allowed:
Notes: The tool will not issue new requests unless there are currently less than or equal to maxQueueSize commands in flight. In conjunction with maxCommandsPerSecond can be used to control impact on the VWS.
Example: maxQueueSize = 10

queueUpdatesToSameObject

Syntax: queueUpdatesToSameObject = value
Description: Controls whether multiple updates to a single business object are done in parallel or serially.
Type: Boolean
Optionality: Optional (default used if not set).
Allowed:
    • true - allow parallel updates
    • false - serial updates only
Default: false
Notes:
Example: queueUpdatesToSameObject = true

Syntax: notificationInterval = value
Description: The number of seconds between reporting progress status to the log file.
Type: Integer
Optionality: Mandatory
Allowed:
Default:
Notes:
Example: notificationInterval = 6

Syntax: maxCommandsPerSecond = value
Description: The maximum number of commands allowed in flight per second.
Type: Integer
Optionality: Mandatory
Allowed:
Default:
Notes:
Example: maxCommandsPerSecond = 10

Syntax: pollTimeUsecs = value
Description: The number of microseconds to wait for a message from a Billing Engine client before polling
Type: Integer
Optionality: Mandatory
Default: 1000000 (one second)
Example: pollTimeUsecs = 100000

Example of VWS correction tool section

This is an example of the eserv.config file correction tool section.

ccsVWSCorrection = {
    ClientIF = {
        clientName = "nccdemo-dev-ccsVWSCorrection"
        heartbeatPeriod = 10000000
        connectionRetryTime = 2
        plugins = []
        billingEngines = [
            {id = 1,
                primary = { ip="PRIMARY_BE_IP", port=1500 },
                secondary = { ip="SECONDARY_BE_IP", port=1500 }
            }
        ]
    }
    maxQueueSize = 10
    queueUpdatesToSameObject = false
    notificationInterval = 6
    maxCommandsPerSecond = 10
    pollTimeUsecs = 100000
} # CCS.ccsVWSCorrection section
Command line parameters

The commands that can be used are:

- **delete_balance**: This deletes the balance from the wallet ID and balance type ID.
- **delete_bucket**: This updates the supplied bucket ID by zeroing the current value of the bucket. The mechanism to physically delete the bucket is up to other (pre-existing) configuration on the VWS as to whether or not zero value buckets are retained or deleted.
- **update_balance**: This updates the supplied balance fields with the new values.
- **update_bucket**: This updates supplied bucket fields with the new values
- **update_wallet**: This updates the supplied wallet fields with the new values.

See Command line examples.

Update balance parameters

The **update_balance**: command has the ability to modify the following fields to schema and business rule acceptable values against a specified balance for a specified wallet:

- **limit_type**
- **minimum_credit**

The balance and wallet key data is supplied in these fields:

- **wallet_id**
- **balance_type**

An example of the update_balance command is:

```
update_balance:wallet_id=4,balance_type=9,limit_type=LCRD,minimum_credit=88881000
```

Update bucket parameters

The **update_bucket**: command has the ability to modify the following fields to schema and business rule acceptable values against a specified bucket for a specified balance and wallet:

- **expiry**
- **value**
- **value_delta**
- **reference**
- **start_date**
- **last_use**
- **neverExpires**
- **never_used**

The balance and wallet key data is supplied in these fields:

- **wallet_id**
- **balance_type**
- **bucket_id**

Examples of the update_bucket command is:
Update wallet parameters

The `update_wallet:` command has the ability to modify the following fields to schema and business rule acceptable values against a specified wallet:

- max_concurrent_state
- never_expired
- expiry
- never_activated
- activation_date
- state

The wallet key data is supplied in this field:

- wallet_id

Examples of the `update_bucket` command is:

```
update_wallet:wallet_id=4,never_expired=true,expiry=20110101115600
update_wallet:wallet_id=47,state=ACTV
```

Delete balance parameters

The `delete_balance:` command deletes the balance.

The balance key data is supplied in these fields:

- wallet_id
- balance_type

Example of the `delete_balance` command is:

```
delete_balance:wallet_id=4,balance_type=99
```

Delete bucket parameters

The `delete_bucket:` command modifies the bucket value field to zero, allowing the VWS to retain or delete the bucket.

The bucket key data is supplied in these fields:

- wallet_id
- balance_type
- bucket_id

Example of the `delete_bucket` command is:

```
delete_bucket:wallet_id=44,balance_type=13,bucket_id=30
```

Command line examples

The commands can be run singularly by typing in at the command line prompt, or as a batch in a file.

This is an example of a file of commands that will do a set of updates (picture a file with 450 lines of the following) that generated the Progress reporting and Audit reporting examples.

```
update_bucket:wallet_id=4,balance_type=9,bucket_id=2,value=5,value_delta=1
update_bucket:wallet_id=4,balance_type=9,bucket_id=2,value=5,value_delta=1
```
update_bucket: wallet_id=4, balance_type=9, bucket_id=2, value=5, value_delta=1
Other examples are:

- update_wallet: wallet_id=4, never_expires=true
- update_balance: wallet_id=4, balance_type=9, limit_type=LCRD
- delete_balance: wallet_id=44, balance_type=9
- delete_bucket: wallet_id=4, balance_type=13, bucket_id=30

**Progress reporting**

The tool sends report information to the logfile, including, the parameters at the start, the status periodically as it executes, and the details of each command processed.

Here is an example showing the expected reporting. The tool is executing a series of commands that add 5c to a particular bucket repeatedly (450 times).

To see the report, on the command line type (for example):

```
-bash-3.00$ ./ccsVWSCorrection -i commandFile -o logFile
```

The logfile report will look something like this:

```
Aug  2 14:54:37.208548 ccsVWSCorrection(29583) NOTICE: ccsVWSCorrection processing starting
Aug  2 14:54:37.213175 ccsVWSCorrection(29583) NOTICE: Connection to BE 1:192.168.10.217-1500 is established.
Aug  2 14:54:37.547017 ccsVWSCorrection(29583) NOTICE: ccsVWSCorrection Tool status: processed 0 of 450 commands: 0.0% complete
Aug  2 14:54:43.091905 ccsVWSCorrection(29583) NOTICE: ccsVWSCorrection Tool status: processed 30 of 450 commands: 6.7% complete
Aug  2 14:54:49.107811 ccsVWSCorrection(29583) NOTICE: ccsVWSCorrection Tool status: processed 72 of 450 commands: 16.0% complete
Aug  2 14:54:55.031967 ccsVWSCorrection(29583) NOTICE: ccsVWSCorrection Tool status: processed 106 of 450 commands: 23.6% complete
Aug  2 14:55:01.058072 ccsVWSCorrection(29583) NOTICE: ccsVWSCorrection Tool status: processed 145 of 450 commands: 32.2% complete
Aug  2 14:55:07.002602 ccsVWSCorrection(29583) NOTICE: ccsVWSCorrection Tool status: processed 179 of 450 commands: 39.8% complete
Aug  2 14:55:13.107238 ccsVWSCorrection(29583) NOTICE: ccsVWSCorrection Tool status: processed 204 of 450 commands: 45.3% complete
Aug  2 14:55:19.081310 ccsVWSCorrection(29583) NOTICE: ccsVWSCorrection Tool status: processed 238 of 450 commands: 52.9% complete
Aug  2 14:55:25.046720 ccsVWSCorrection(29583) NOTICE: ccsVWSCorrection Tool status: processed 278 of 450 commands: 61.8% complete
Aug  2 14:55:31.141610 ccsVWSCorrection(29583) NOTICE: ccsVWSCorrection Tool status: processed 310 of 450 commands: 68.9% complete
Aug  2 14:55:37.082081 ccsVWSCorrection(29583) NOTICE: ccsVWSCorrection Tool status: processed 346 of 450 commands: 76.9% complete
Aug  2 14:55:43.022000 ccsVWSCorrection(29583) NOTICE: ccsVWSCorrection Tool status: processed 386 of 450 commands: 85.8% complete
Aug  2 14:55:49.096070 ccsVWSCorrection(29583) NOTICE: ccsVWSCorrection Tool status: processed 418 of 450 commands: 92.9% complete
Aug  2 14:55:54.407038 ccsVWSCorrection(29583) NOTICE: ccsVWSCorrection Tool status: processed 450 of 450 commands: 100.0% complete
Aug  2 14:55:54.407308 ccsVWSCorrection(29583) NOTICE: ccsVWSCorrection processing complete
```
Audit reporting

The audit log contains structured fields (keyed by command number, time stamp and log record type. This is to permit convenient grepping, filtering, sorting and analysis of the log records after the run.

A log record can be of type:

- **COMMAND**: dumping the command being called
- **INFO**: displaying any informational message
- **WARNING**: displaying some warning condition
- **ERROR**: displaying an error in order to explain why the command did not run
- **AUDIT**: for commands that got as far as an update request, one or more of these show what fields were modified. Commands that delete business objects will display the current value of that object and any children it contains, to assist with recovery should it be necessary.

This is the log from the Command line examples and Progress reporting examples.

```
00000001 [20110802145437.549545] COMMAND:
update_bucket:wallet_id=4,balance_type=9,bucket_id=2,value=5,value_delta=1
00000002 [20110802145437.652071] COMMAND:
update_bucket:wallet_id=4,balance_type=9,bucket_id=2,value=5,value_delta=1
00000002 [20110802145437.652117] WARNING: We're already executing a command for wallet ID 4 (we'll retry shortly.)
00000001 [20110802145437.671977] AUDIT : update_bucket: wallet_id=4,balance_type=9,bucket_id=2,old_value=6330,new_value=6335,old_reference=,new_reference=,old_start_date=19700101000000,new_start_date=19700101000000,old_never_expires=1,new_never_expires=1,old_expiry=19700101000000,new_expiry=19700101000000,old_never_used=0,new_never_used=0,old_last_use=20110802025232,new_last_use=20110802025232
00000003 [20110802145437.882058] COMMAND:
update_bucket:wallet_id=4,balance_type=9,bucket_id=2,value=5,value_delta=1
00000003 [20110802145437.882146] WARNING: We're already executing a command for wallet ID 4 (we'll retry shortly.)
00000003 [20110802145437.990985] WARNING: We're already executing a command for wallet ID 4 (we'll retry shortly.)
00000002 [20110802145438.045385] AUDIT : update_bucket: wallet_id=4,balance_type=9,bucket_id=2,old_value=6335,new_value=6340,old_reference=,new_reference=,old_start_date=19700101000000,new_start_date=19700101000000,old_never_expires=1,new_never_expires=1,old_expiry=19700101000000,new_expiry=19700101000000,old_never_used=0,new_never_used=0,old_last_use=20110802025437,new_last_use=20110802025437
00000003 [20110802145438.172316] AUDIT : update_bucket: wallet_id=4,balance_type=9,bucket_id=2,old_value=6340,new_value=6345,old_reference=,new_reference=,old_start_date=19700101000000,new_start_date=19700101000000,old_never_expires=1,new_never_expires=1,old_expiry=19700101000000,new_expiry=19700101000000,old_never_used=0,new_never_used=0,old_last_use=20110802025438,new_last_use=20110802025438
```

Lots of lines removed for conciseness.

```
00000449 [20110802145553.946776] COMMAND:
update_bucket:wallet_id=4,balance_type=9,bucket_id=2,value=5,value_delta=1
00000449 [20110802145554.013305] AUDIT : update_bucket: wallet_id=4,balance_type=9,bucket_id=2,old_value=6350,old_reference=,new_reference=,old_start_date=19700101000000,new_start_date=19700101000000,old_never_expires=1,new_never_expires=1,old_expiry=19700101000000,new_expiry=19700101000000,old_never_used=0,new_never_used=0,old_last_use=20110802025553,new_last_use=20110802025553
00000450 [20110802145554.122104] WARNING: We're already executing a command for wallet ID 4 (we'll retry shortly.)
00000450 [20110802145554.295870] AUDIT : update_bucket: wallet_id=4,balance_type=9,bucket_id=2,old_value=6350,new_value=6355,old_reference=,new_reference=,old_start_date=19700101000000,new_start_date=19700101000000,old_never_expires=1,new_never_expires=1,old_expiry=19700101000000,new_expiry=19700101000000,old_never_used=0,new_never_used=0,old_last_use=20110802025554,new_last_use=20110802025554
```

Statistics:
- Completed commands = 450
- Information acks received = 450
- Primary information acks received = 450
- Primary update acks received = 450
- Total commands = 450
Update acks received = 450
Wallet Info Requests sent to Primary BE = 450

ccsVNSCorrection stopped at Tue Aug  2 14:55:54 2011

The log report shows that all commands were (eventually, there were some cases of the tool waiting for a previous update for the same wallet ID to finish, but this is normal in a file with multiple commands against the same wallet) successful, and the bucket value grew from an initial balance of 6330 (old value on first audit record) to 8580 (new value on last audit record). The difference is 450 x 5, so all updates were applied correctly.
Overview

Introduction

This chapter explains the important processes on each of the server components in the NCC, and a number of example troubleshooting methods which will help aid the troubleshooting process before raising a support ticket.

In this chapter

This chapter contains the following topics.

Common Troubleshooting Procedures 127
Possible Problems 127
Process Failure Recovery 129

Common Troubleshooting Procedures

Introduction

Refer to NCC System Administrator's Guide for troubleshooting procedures common to all NCC components.

Possible Problems

Introduction

This topic lists common problems and actions which can be taken to investigate or solve them. This list enables you to check for alarms based on the overall behavior you are experiencing.

Database failure

Upon network failure, any request or response may be lost. Pending Database (DB) write and EDRs will be lost.

Failure scenarios

This table lists a range of failure scenarios and a description of the events that will happen as a result.

For more information about resynchronization, see Resynchronizations (on page 16).

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Resulting Events</th>
</tr>
</thead>
<tbody>
<tr>
<td>VWS is running. beServer core dumps, losing all contexts and BeClient connections.</td>
<td>1 beServer recovers, finds the current state from beVWARS, determines that the VWS should be in Recovery state and makes it so. 2 beServer prepares to receive contexts from beServer on the other</td>
</tr>
<tr>
<td></td>
<td>VWS.</td>
</tr>
<tr>
<td>---</td>
<td>------</td>
</tr>
<tr>
<td>3</td>
<td>beGroveller detects the dropped connection and a failover is triggered.</td>
</tr>
<tr>
<td>4</td>
<td>beSync starts attempting recovery.</td>
</tr>
<tr>
<td>5</td>
<td>Remote beServer starts sending contexts to the local beServer (and receives Operations from beSync).</td>
</tr>
<tr>
<td>6</td>
<td>beSync should complete its recovery quickly (it should have already been in Sync). beSync will tell beVWARS to move to Running state, when the beVWARS has finished sending contexts it will move to Running.</td>
</tr>
<tr>
<td>7</td>
<td>beServer starts accepting connections from BeClient processes.</td>
</tr>
<tr>
<td>8</td>
<td>beGroveller establishes connection with beServer and starts grovelling.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>beVWARS core dumps, loses all reservations and cached wallets, then restarts.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>beServer and beSync are informed that the beVWARS process had died.</td>
</tr>
<tr>
<td>2</td>
<td>beServer closes all open connections as nicely as it can.</td>
</tr>
<tr>
<td>3</td>
<td>beVWARS restarts, prepares to send contexts to beServer (and receive Operations from beSync).</td>
</tr>
<tr>
<td>4</td>
<td>beServer and beSync recognise beVWARS recovery.</td>
</tr>
<tr>
<td>5</td>
<td>beSync initiates recovery (includes getting all reservation details from other VWS).</td>
</tr>
<tr>
<td>6</td>
<td>beGroveller detects the dropped connection and a failover is triggered.</td>
</tr>
<tr>
<td>7</td>
<td>When beSync completes recovery, the process completes as above (in beServer failure).</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>VWS state is running. beSync core dumps and restarts.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>beSync restarts, gets current status from beVWARS.</td>
</tr>
<tr>
<td>2</td>
<td>If the beVWARS is in Recovery or Running state it starts recovery.</td>
</tr>
<tr>
<td>3</td>
<td>beSync processes as normal, but does not force the system into Recovery (which would deny connections, and this is not required) groveller proceeds as it was.</td>
</tr>
<tr>
<td>4</td>
<td>beSync proceeds, when the inSync threshold is reached it tells the beVWARS to go to Running state (which it may already be in).</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Primary VWS has power turned off. BeClient detects failure of primary VWS.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Primary VWS is turned back on.</td>
</tr>
<tr>
<td>2</td>
<td>SLEE starts up, all SLEE processes start.</td>
</tr>
<tr>
<td>3</td>
<td>beServer starts disabled, refuses BeClient connections.</td>
</tr>
<tr>
<td>4</td>
<td>beGroveller attempts to connect to beServer and fails. It doesn't start processing.</td>
</tr>
<tr>
<td>5</td>
<td>beSync starts, reads in existing sync file repository. At this point beSync source will not accept connections from the remote beSync sink as we do not want to send anything.</td>
</tr>
<tr>
<td>6</td>
<td>BeClient swaps to sending messages to Secondary VWS, resending any it does not have responses for (and marks them as duplicates).</td>
</tr>
<tr>
<td>7</td>
<td>beSync asks each local beVWARS for their last written Sequence Number.</td>
</tr>
<tr>
<td>8</td>
<td>beSync looks to see if it needs to process files locally to write updates that are in the sync files but not in the database. This is done by looking to see if there is any later sequence numbers in the files.</td>
</tr>
<tr>
<td>9</td>
<td>If later sequence numbers are found, they are read and sent to the beVWARS.</td>
</tr>
<tr>
<td>10</td>
<td>While this is proceeding, the beSync source will start accepting</td>
</tr>
</tbody>
</table>
connections from the remote beSync sink.

11 When all local updates have been performed, beSync sink requests all reservations from the remote VWS to populate the beVWARS.

Note: This is an extra step only performed on full recovery.

12 The local beSync sink establishes a link to the remote VWS and requests all updates since the last remote update we have recorded on our database.

13 Updates stream to us, and we confirm them in chunks.

14 The timestamp on every update is checked against the wall clock. If the difference is less than the (configurable) inSyncThreshold then we consider ourselves to be inSync and tell beServer to start accepting connections again.

15 beServer starts accepting connections again.

16 beGroveller establishes connection with beServer and starts grovelling.

For more information about beGroveller failover, see beGroveller quorum (on page 130).

Process Failure Recovery

Startup checks

On startup, or failure (and restart) of the beServer or beVWARS, we must get all reservations and server contexts from the peer VWS. Both the beServer and beVWARS must be present if this is to be successful.

Startup process

Once you have started the SLEE, the following events will occur.

<table>
<thead>
<tr>
<th>Stage</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>All processes startup in disabled/startup state.</td>
</tr>
<tr>
<td>2</td>
<td>beSync waits until it can contact beVWARS.</td>
</tr>
<tr>
<td>3</td>
<td>Once beSync can, it starts local recovery, by sending updates to the beVWARS.</td>
</tr>
<tr>
<td>4</td>
<td>The local beSync establishes a connection to the beSync on the remote VWS, and asks for contexts and reservations.</td>
</tr>
<tr>
<td>5</td>
<td>Once the local beSync has all the remote contexts and updates, it requests the remote updates from beSync.</td>
</tr>
<tr>
<td>6</td>
<td>Once remote updates are within a couple of seconds of the current time, beSync tells the beVWARS to change to running state.</td>
</tr>
<tr>
<td>7</td>
<td>beVWARS passes the state change message to beServer.</td>
</tr>
<tr>
<td>8</td>
<td>Once beServer has Running messages from all beVWARS, it goes into running mode, and opens for client connections.</td>
</tr>
<tr>
<td>9</td>
<td>If grovelling is available at this time, beGroveller can now grovel, and will respond to requests for more wallets to grovel from each of the beVWARS processes.</td>
</tr>
</tbody>
</table>

Restarts while in state Recovery

If you restart while in state recovery, the following events will occur.
beSync:
1. Queries the beVWARS, finds it is recovering, and starts requesting everything from the other beSync again.
2. Reservations in the beVWARS are overwritten.
3. Contexts in the beServer are overwritten.

beVWARS:
1. Comes up in disabled state.
2. beServer gets a dialog closed event and disables itself, then it tries to contact beVWARS to put it in Recovery mode.

beServer:
1. Same as SLEE startup; contacts the beVWARS and resets Recovery mode.

**Restarts while in state Running**

If you restart while in state running, the following events will occur.

- beSync queries the beVWARS and starts up running.
- beVWARS comes up disabled.
- beServer will get a dialog closed, disable itself, and then same as SLEE startup. Then it will follow the same process as a SLEE restart.
- beGroveller will start disabled, and will start processing when beServer starts accepting connections.
- We have lost all of the updates between the committed database's sequence number and those in the beVWARS, however the sync files still record all of these (they have not been removed as beSync hasn't received a COMMIT message yet).
- Each beVWARS clears its cache, reads the local and remote sequence number from the database, goes into recovery mode - broadcasting these SSEQ numbers.
- beVWARS will then ignore all operations from beSync until it sees one with the SessionNumber set to the ID of the control message it sent to set the VWS state to Recovery - beSync may have operations queued on the beVWARS with SSEQs AFTER those of the database (we do not want to skip those in between).
- beSync sees these new SSEQ numbers and sends all of the local and remote transactions it has to the beVWARS. The first operation message has the SessionNumber set to the ID of the control message that set the state to Recovery.

**beGroveller quorum**

beGroveller is designed to only run on the primary Voucher and Wallet Server in a pair. However, groveling activity will failover to the secondary Voucher and Wallet Server if the Voucher and Wallet Server fails.

beGroveller determines whether it should pass groveling work to beVWARS processes, by checking whether it is on the primary and whether it can connect to the:

- Local beServer process
- beServer on the other VWS in the pair
- SMS specified in `quorumHost` (on page 57)

**Establishing quorum on primary VWS**

beGroveller determines whether it is running on a primary VWS by checking the value of `amPrimary` (on page 35). If `amPrimary` is set to true, the beGroveller is running on a primary VWS.
If the beGroveller on a primary VWS can connect to the local beServer, beGroveller will respond to beVWARS grovel requests with lists of wallet IDs to grovel. If beGroveller cannot connect to the local beServer, it assumes the VWS is disabled or recovering and will not return work to beVWARS processes.

**Establishing quorum on secondary VWS**
beGroveller determines whether it is running on a secondary VWS by checking the value of `amPrimary` (on page 35). If `amPrimary` is set to false, the beGroveller is running on a secondary VWS.

If beGroveller:

- Cannot connect to the local beServer, it assumes the VWS is disabled or recovering and will not return work to beVWARS processes.
- On a secondary VWS can connect to the local beServer, beGroveller will check whether it can connect to the beServer on the other VWS in the pair. If it can connect to the remote beServer, it will assume the primary is running and will not respond to beVWARS grovel requests with lists of wallet IDs to grovel.
- Can connect to the local beServer, but cannot connect to the beServer on the VWS, it will check whether it can ping the remote VWS. If it can ping the remote VWS it assumes the remote VWS is disabled and will start groveling.
- Cannot ping the remote VWS it will attempt to ping the SMS specified in `quorumHost` (on page 57). If it can ping `quorumHost`, it will assume the VWS pair has failed over, and will start to respond to beVWARS requests with lists of wallet IDs to grovel. If it cannot ping `quorumHost`, it will assume it is not on the main network, and will not respond to beVWARS requests with lists of wallet IDs to grovel.
Chapter 6

About Installation and Removal

Overview

Introduction

This chapter provides details of the installation and removal process for the application.

In this chapter

This chapter contains the following topics.

- Installation and Removal Overview
- Configuring for Raw Device Support
- Checking the Installation

Installation and Removal Overview

Introduction

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

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

Voucher & Wallet Server packages

An installation of Voucher & Wallet Server includes the following packages, on the:

- SMS:
  - beSms
- SLC:
  - beScp
- VWS:
  - beBe

Configuring for Raw Device Support

Introduction

This section details how to configure the VWS packages on the Voucher and Wallet Servers for raw device support.

CCS and BE can allocate tablespace storage based on raw (without a file system) partitions in order to enhance the performance of the VWS database.
When using the raw devices option, the raw partitions must be created by the user prior to continuing the installation of the database using tools such as the system’s format command.

The raw devices file (of which the user is prompted to complete before continuing the installation) must contain the full paths of the device files for the appropriate partitions.

**System Device Configuration**

To configure the Voucher and Wallet Server to use raw devices as datafiles, once installation is complete edit the following file:

```
/IN/service_packages/E2BE/db/install/create/BE/scripts/be_system_devices.sh
```

The `be_system_devices.sh` file contains entries for:

- 3 control file datafiles
- 1 system tablespace datafile
- 2 groups of 8 redo log datafiles
- 8 temporary tablespace datafiles
- 1 tools tablespace datafile
- 1 users tablespace datafile
- 5 undo tablespace datafiles
- 4 rollback segment datafiles (which will be unused for a 9i or post-9i database)

**Configuring the file**

Follow these steps to configure the entries in `be_system_devices.sh`.

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
</table>
| 1    | Configure the control file entries.  
Create a raw device of size 301 MB (consult the partitioning guide).  
Append the line `CONTROL_FILE_X=` with the full path of the raw device node, for example:  
`CONTROL_FILE_1=/dev/rdsk/c1t4d3s0` |
| 2    | Configure the system tablespace datafile.  
Create a raw device of size 451 MB (consult the partitioning guide).  
Append the line `SYSTEM_TABLESPACE_DATAFILE=` with the full path of the raw device node, for example:  
`SYSTEM_TABLESPACE_DATAFILE=/dev/rdsk/c1t4d3s0` |
| 3    | Configure the redo logs.  
Create two raw devices. Either:  
- one each of size 11 MB if you wish to use the test sizing, or  
- two raw devices, each of size 41 MB (consult the partitioning guide).  
**Note:** The redo logs are split into two groups of eight. There eight logs each with a backup. Follow these steps for each redo log.  
Append the line `REDO_LOGX_FILEA=` with the full path of the first raw device node, for example:  
`REDO_LOG1_FILEA=/dev/rdsk/c1t4d3s0`  
Append the line `REDO_LOGX_FILEB=` with the full path of the second raw device node, that is:  
`REDO_LOG1_FILEB=/dev/rdsk/c1t4d3s1` |
| 4    | Configure the temporary tablespace datafile entries.  
Create a raw device of size 102 MB if you wish to use the test sizing, otherwise create |
### E2BE_ADMIN Schema Device Configuration

To configure the Voucher and Wallet Server to use Raw Devices as datafiles, once installation is complete edit the following file:

```
/IN/service_packages/E2BE/db/install/create/BE/scripts/autogen_be_devices.sql
```

The `autogen_be_devices.sql` file contains a number of repeated entries of the form:

```sql
/*
# Datafiles for tablespace: BE_DATA
*/
```

```sql
/* Datafile 1 partition size should be 10M + 1M */
define DEVICE_BE_DATA1 =
/* Datafile 2 partition size should be 10M + 1M */
define DEVICE_BE_DATA2 =
```

---

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Create a raw device of size 2042 MB (consult the partitioning guide). Append the line TEMP_DATAFILE1= with the full path of the raw device node, for example: TEMP_DATAFILE1=/dev/rdsk/c1t4d3s0</td>
</tr>
<tr>
<td>2</td>
<td>Configure the tools tablespace datafiles. Create a raw device of size 16 MB (consult the partitioning guide below). Append the line TOOLS_DATAFILE1= with the full path of the raw device node, for example: TOOLS_DATAFILE1=/dev/rdsk/c1t4d3s0</td>
</tr>
<tr>
<td>3</td>
<td>Configuring the users tablespace datafiles. Create a raw device of size 101 MB (consult the partitioning guide). Append the line USERS_DATAFILE1= with the full path of the raw device node, for example: USERS_DATAFILE1=/dev/rdsk/c1t4d3s0</td>
</tr>
<tr>
<td>4</td>
<td>Configure the undo tablespace datafiles. Create a raw device of size 201 MB if you wish to use the test sizing, otherwise create a raw device of size 1991 MB (consult the partitioning guide). Note: If installing on a pre-9i system, these entries should be skipped. Append the line UNDOTBS1_DATAFILE1= with the full path of the raw device node, for example: UNDOTBS1_DATAFILE1=/dev/rdsk/c1t4d3s0</td>
</tr>
<tr>
<td>5</td>
<td>Configure the rollback segment datafiles. Create a raw device of size 102 MB if you wish to use the test sizing, otherwise create a raw device of size 2042 MB (consult the partitioning guide). Note: If installing on a post-9i system, these entries should be skipped. Append the line RBS_DATAFILE1= with the full path of the raw device node, for example: RBS_DATAFILE1=/dev/rdsk/c1t4d3s0</td>
</tr>
<tr>
<td>6</td>
<td>Save and backup the newly populated device file (as it may be removed by the uninstallation process later).</td>
</tr>
</tbody>
</table>
/*
#*******************************************************************************
*/

Note: The comment for each datafile listed per tablespace, contains a size in mb of the form XM + 1M, for example:

/* Datafile 2 partition size should be 10M + 1M */

Configuring the file

Follow these steps to configure the entries in autogen_be_devices.sql.

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
</table>
| 1    | Edit the file:  
       /IN/service_packages/E2BE/db/install/create/BE/scripts/autogen_be_devices.sql |
| 2    | For each individual datafile listed create a raw device of the size specified in the comment (consult the partitioning guide).  
       For the above example create two raw devices, each of 11 mb. |
| 3    | For each created datafile append the line of the form DEVICE_BE_DATA{X}= with the full path of the new device node  
       Example:  
       DEVICE_BE_DATA1=/dev/rdsk/c1t4d3s0  
       DEVICE_BE_DATA2=/dev/rdsk/ct14d3s1 |
| 4    | Save and backup the newly populated device file (as it may be removed by the un-installation process later) |

Raw Device Setup

Raw devices can be created on Solaris 9 using the format command (for additional information, run the command 'man format') as follows:

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
</table>
| 1    | Start the format program by running the command ‘format’.  
       Result: You will be presented with a list of disks. |
| 2    | Specify a disk to place the new partition. Type the option number of the applicable disk, and press Enter.  
       Result: You will be presented with a list of format options. |
| 3    | Define a partition table for the applicable disk. Type partition and press Enter.  
       Result: You will be presented with a list of 8 partitions (labeled 0 – 7), as well as some supporting options. |
| 4    | Type print and press Enter to list the current partition table for this disk.  
       Result: Details of each currently defined partition (and those that are not defined) are displayed on the screen. |
| 5    | Choose a partition that is not defined already (one that has a size of 0 defined). Type the partition number and press Enter.  
       Result: You will be asked to enter the partition id tag. |
| 6    | Press Enter.  
       Result: You will be asked for the partition permissions option. |
| 7    | Type wm and press Enter. |
Step | Action |
--- | --- |
| **Result:** You will be asked for the starting cylinder of the partition. | |
| 8 | If this is not the first partition, choose a cylinder that follows on from the last defined partition. For example: if there is a last partition defined between cylinders 1-1024, choose the cylinder 1025 as the starting cylinder. |
| 9 | Each partition will have a corresponding file system node in the directory `/dev/rdsk`. The device nodes for the new partitions will start with the name of the disk, and end with the number of the partition. For example: if c1t1d3 is the disk, and the partition is number 3, then the device file will be `/dev/rdsk/c1t1d3s3`. The above device file will be a soft link. Please ensure that the linked to file is readable and writable by the Oracle user. |

### Checking the Installation

**Introduction**

Refer to this checklist to ensure that VWS has installed correctly.

The end of the package installation process specifies a script designed to check the installation just performed. They must be run from the command line.

**Checklist**

Follow these steps in this checklist to ensure VWS has been installed on an VWS machine correctly.

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Log into the VWS machine as root.</td>
</tr>
<tr>
<td>2</td>
<td>Check the following directory structure exists with the subdirectory: <code>/IN/service_packages/E2BE</code></td>
</tr>
<tr>
<td>3</td>
<td>Check the directory contains subdirectories and that all are owned by: ebe_oper user (group esg)</td>
</tr>
<tr>
<td>4</td>
<td>Log into the system as ebe_oper.</td>
</tr>
<tr>
<td><strong>Note:</strong> This step is to check that the ebe_oper user is valid.</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Type <code>sqlplus /</code> No password is required.</td>
</tr>
<tr>
<td><strong>Note:</strong> This step is to check that the ebe_oper user has valid access to the database.</td>
<td></td>
</tr>
<tr>
<td>Ensure that the ORACLE_SID is set.</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Ensure that VWS and CCS tables have been added to the database.</td>
</tr>
<tr>
<td>7</td>
<td>Check the entries of the following file: <code>/etc/inittab</code></td>
</tr>
<tr>
<td><strong>Inittab Entries Reserved for VWS on VWS:</strong></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td><code>be_1</code> <code>bin/beCDRMoverStartup.sh</code> (Runs beCDRMoverStartup, which moves completed EDR files into an output directory for later processing.)</td>
</tr>
<tr>
<td>2</td>
<td><code>be_2</code> <code>bin/cmnPushFilesStartup.sh</code> (Runs cmnPushFiles, which moves the EDRs to a configured destination machine)</td>
</tr>
<tr>
<td>Step</td>
<td>Action</td>
</tr>
<tr>
<td>------</td>
<td>--------</td>
</tr>
<tr>
<td></td>
<td>(usually the SMS).</td>
</tr>
</tbody>
</table>
NCC Glossary of Terms

AAA

ACS
Advanced Control Services configuration platform.

ANI
Automatic Number Identification - Term used in the USA by long-distance carriers for CLI.

API
Application Programming Interface

BE
Billing Engine

BFT
Billing Failure Treatment - the process that is applied if the system has lost all connections to a billing engine. It allows for limited continuation of call processing functions, if configured.

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

CDR
Call Data Record

Note: The industry standard for CDR is EDR (Event Detail Record). Over time EDR will replace CDR in the Oracle documentation.

CLI
Calling Line Identification - the telephone number of the caller. Also referred to as ANI.

Connection
Transport level link between two peers, providing for multiple sessions.

CORBA
Common Object Request Broker Architecture. It is a framework that provides interoperability between objects built in different programming languages, running on different physical machines perhaps on different networks. It specifies an Interface Definition Language, and API that allows client / server interaction with the ORB.
CPU
Central Processing Unit

cron
Unix utility for scheduling tasks.

crontab
File used by cron.

DB
Database

Diameter
A feature rich AAA protocol. Utilises SCTP and TCP transports.

DTMF
Dual Tone Multi-Frequency - system used by touch tone telephones where one high and one low frequency, or tone, is assigned to each touch tone button on the phone.

E2BE
Code used to designate some components and path locations used by the UBE.

EDR
Event Detail Record

Note: Previously CDR. The industry standard for CDR is EDR (Event Detail Record). Over time EDR will replace CDR in the NCC documentation.

FOX
Fast OSA eXtensions. A TCP/IP billing protocol intended for use with external vendors. Based on OSA, it fills in functional gaps missing in OSA, and defines “combined” OSA operations to increase platform throughput. Uses a non-CORBA transport layer in order to provide enhanced fail-over and connection redundancy.

GPRS
General Packet Radio Service - employed to connect mobile cellular users to PDN (Public Data Network- for example the Internet).

GSM
Global System for Mobile communication.

It is a second generation cellular telecommunication system. Unlike first generation systems, GSM is digital and thus introduced greater enhancements such as security, capacity, quality and the ability to support integrated services.
HLR

The Home Location Register is a database within the HPLMN (Home Public Land Mobile Network). It provides routing information for MT calls and SMS. It is also responsible for the maintenance of user subscription information. This is distributed to the relevant VLR, or SGSN (Serving GPRS Support Node) through the attach process and mobility management procedures such as Location Area and Routing Area updates.

HPLMN

Home PLMN

HTML

HyperText Markup Language, a small application of SGML used on the World Wide Web. It defines a very simple class of report-style documents, with section headings, paragraphs, lists, tables, and illustrations, with a few informational and presentational items, and some hypertext and multimedia.

IN

Intelligent Network

IP

1) Internet Protocol
2) Intelligent Peripheral - This is a node in an Intelligent Network containing a Specialized Resource Function (SRF).

IP address

Internet Protocol Address - network address of a card on a computer

IPC

Interprocess Communication

MAP

Mobile Application Part - a protocol which enables real time communication between nodes in a mobile cellular network. A typical usage of the protocol would be for the transfer of location information from the VLR to the HLR.

Messaging Manager

The Messaging Manager service and the Short Message Service components of Oracle Communications Network Charging and Control product. Component acronym is MM (formerly MMX).

MM

Messaging Manager. Formerly MMX, see also XMS (on page 144) and Messaging Manager (on page 141).

MSC

Mobile Switching Centre. Also known as a switch.
MT
Mobile Terminated

Oracle
Oracle Corporation

ORB
Object Request Broker. Within an Object based communication system, an ORB keeps track of the actual addresses of all defined objects and thus is used to route traffic to the correct destination. The CORBA defines the ORB in a series of standards enabling different platforms to share common information.

OSA
Open Service Access provides a standard interface through which developers can design services that may interact with functions within the network.

PI
Provisioning Interface - used for bulk database updates/configuration instead of GUI based configuration.

PIN
Personal Identification Number

PLMN
Public Land Mobile Network

SCTP
Stream Control Transmission Protocol. A transport-layer protocol analogous to the TCP or User Datagram Protocol (UDP). SCTP provides some similar services as TCP (reliable, in-sequence transport of messages with congestion control) but adds high availability.

Session
Diameter exchange relating to a particular user or subscriber access to a provided service (for example, a telephone call).

SGML

SGSN
Serving GPRS Support Node

SLC
Service Logic Controller (formerly UAS).
SLEE
Service Logic Execution Environment

SMS
Depending on context, can be:
- Short Message Service
- Service Management System platform
- NCC Service Management System application

SRF
Specialized Resource Function - This is a node on an IN which can connect to both the SSP and the SLC and delivers additional special resources into the call, mostly related to voice data, for example play voice announcements or collect DTMF tones from the user. Can be present on an SSP or an Intelligent Peripheral (IP).

SSP
Service Switching Point

System Administrator
The person(s) responsible for the overall set-up and maintenance of the IN.

TCAP
Transaction Capabilities Application Part – layer in protocol stack, message protocol.

TCP
Transmission Control Protocol. This is a reliable octet streaming protocol used by the majority of applications on the Internet. It provides a connection-oriented, full-duplex, point to point service between hosts.

Telco
Telecommunications Provider. This is the company that provides the telephone service to customers.

Telecommunications Provider
See Telco.

URL
Uniform Resource Locator. A standard way of specifying the location of an object, typically a web page, on the Internet.

USSD
Unstructured Supplementary Service Data - a feature in the GSM MAP protocol that can be used to provide subscriber functions such as Balance Query and Friends and Family Access.
**VLR**
Visitor Location Register - contains all subscriber data required for call handling and mobility management for mobile subscribers currently located in the area controlled by the VLR.

**VWS**
Oracle Voucher and Wallet Server (formerly UBE).

**XML**
eXtensible Markup Language. It is designed to improve the functionality of the Web by providing more flexible and adaptable information identification.

It is called extensible because it is not a fixed format like HTML. XML is a `metalam` - a language for describing other languages—which lets you design your own customized markup languages for limitless different types of documents. XML can do this because it's written in SGML.

**XMS**
Three letter code used to designate some components and path locations used by the Oracle Communications Network Charging and Control Messaging Manager (on page 141) service and the Short Message Service. The published code is MM (on page 141) (formerly MMX).
Deprecated SLEE Name Definitions • 40
destinationDirectory • 44
Diameter • 146
directMaxDelaySeconds • 96
Disabled • 22
Document Conventions • vi
downstreamOverloadSleepUSec • 65, 74, 92
DTMF • 146
duplicateDetection parameters • 96

E
E2BE • 146
E2BE_ADMIN Schema Device Configuration • 141
Editing the file • 31
edr • 84
EDR • 146
EDR Processing • 9, 24
EDR processing diagram • 24
EDR triggers • 25
enabled • 107
enableGrovelling • 37, 56, 98
endsAt • 59
Error and throttle flow • 118
eserv.config Configuration • 29, 30
eserv.config files delivered • 31
eserv.config parameters • 124
Establishing quorum on primary VWS • 137
Establishing quorum on secondary VWS • 137
Event Storage Interface • 117
Event Storage SLEE Events • 119
Example • 79
Example BE shared parameters configuration • 36
Example configuration • 45, 54, 56, 71, 78, 107, 112, 114, 116
Example eserv.config configuration • 31
Example of VWS correction tool section • 126
expectedKeep • 68
eXireAtMidnightTZ • 20, 110
eXireBucketsForExpiredWallets • 21, 111, 112
eXireNegativeBuckets • 21, 111
Expiry diagram • 20

F
-F • 116
Failure • 46, 112, 117
Failure scenarios • 18, 133
failureRetryTime • 84
filledBufferThreshold • 57
Flow 1 • 88
Flow 10 • 89
Flow 2 • 89
Flow 3 • 89
Flow 4 • 89
Flow 5 • 89
Flow 6 • 89
Flow 7 • 89
Flow 8 • 89
Flow 9 • 89
Flush process • 13
flushPeriod • 104
FOX • 146
freeDiskSpaceCheckInterval • 39, 40
freeDiskSpaceShutdownThres • 39, 40
freeDiskSpaceWarningThres • 39, 40
Functions • 1

G
GPRS • 146
groveller parameters • 56, 97
grovellerIfNamePrefix • 40
GSM • 146

H
-h • 116
handlers • 8, 61, 65, 90, 92, 109, 110
heartbeatPeriod • 50, 58, 120, 124
heartbeatPeriodSeconds • 75
HLR • 147
HPLMN • 147
HTML • 147

I
id • 48
IN • 147
Installation and Removal Overview • 139
inSyncReportingPeriodRecords • 76
inSyncThresholdSeconds • 75
Introduction • 1, 18, 21, 24, 25, 27, 29, 30, 41, 133, 139, 143
Introduction to VWS • 1
ip • 48
IP • 147
IP address • 147
IPC • 147

K
keepDirectSeconds • 96
keepSyncSeconds • 96

L
libBeClientIF • 8, 120
libbeEventFactory • 7, 121
libbeMsgRouterDefault • 61, 66, 119
libclientBcast • 49, 53, 120
listenInterface • 77
listenPort • 77
Loading eserv.config changes • 31
Loading SLEE.cfg changes • 42
localUpdateChunkSize • 76
lowDiskSpaceNotificationInterval • 40
ludProcessingTime • 58

146 NCC Voucher and Wallet Server Technical Guide
Main components • 2
Main components diagram • 2
MAP • 147
maxBuckets • 113
maxConnections • 85
maxDownstreamQueueLength • 74, 93
maxDownstreamQueueLength • 66
maxIDPerResponse • 58, 99
maxLoopSize • 100, 101
maxOpenDialogTime • 93
maxOutstandingMessages • 51, 120
maxQueueLength • 74, 78
maxQueueSize • 125
maxRatePerUAS • 85
maxRecordsPerFile • 102
maxRetriesBeforeSeconds • 76
maxSendReservationsToSync • 93
maxSize • 101, 102
maxTransactionsPerSet • 94
Merging wallets processes • 11
messageRoutingPlugins • 8, 61, 66, 67, 119
messageTimeoutSeconds • 51, 120
Messaging Manager • 147, 150
MFile data types • 25
MFile Updates • 25
Migrating wallets • 5
Migrating wallets • 5
minResyncReservationLength • 94
MM • 147, 150
MSC • 147
msgRouterDefault • 67, 120
MT • 148

noExpectedKeep • 69
noProcessingTimes • 59
notEndActions • 52, 68, 120
Notification flows • 88
Notification overview • 86
Notification requests • 86
noWorkSleepTime • 74
numberOfRecordsToCommit • 44

operation • 83
operationSet • 83
Oracle • 148
Oracle environment variables • 30
oraclePassword • 44
oracleService • 45
oracleUser • 45
oracleUserAndPassword • 38
ORB • 148
OSA • 148
OSD Parameters • 82

osd_scps • 83
outDirectory • 45
Output • 46, 55, 72, 79, 86, 109, 113, 117
Overview • 1, 29, 43, 117, 133, 139

Parameters • 37, 44, 47, 57, 63, 73, 92, 110, 113, 115
Password • 82
peerDatabaseLogin • 97
peerWalletCheckRetrySeconds • 97
periodMsec • 98
PI • 148
PIN • 148
PLMN • 148
plugins • 8, 9, 52, 90, 94, 120, 121, 124
Plug-ins • 61, 90, 110
port • 49
Possible Problems • 133
Prerequisites • v
primary • 48
primaryFailbackInterval • 53
Process • 55, 80
Process Failure Recovery • 66, 70, 135
processExpiredBuckets • 60
Progress reporting • 129
purge • 68
purgeInterval • 69
Purpose • 36, 43, 46, 55, 61, 72, 79, 89, 109, 110, 113, 114, 119, 120, 121
Purpose of tool • 123

queueUpdatesToSameObject • 125
Queuing and flushing updates • 9, 13, 25
quiesceLength • 65
quorumHost • 60, 137

Raw Device Setup • 142
recordSendingChunkSize • 78
Recovering • 22
recoveryReportInterval • 70
Related documents • v
remoteBEhostname • 76
remoteBEPort • 77
removeEmptyBuckets • 21, 112
reportPeriodSeconds • 53, 120
Request handling • 7
Request processing • 7
requestHighWaterMark • 98
requestTimeout • 99
reservationExpiryCheckMilliseconds • 95
Reservations and billing diagram • 7
responseTag • 85
Restarts while in state Recovery • 136
Restarts while in state Running • 136
Resynchronizations • 17, 22, 133
retryConnectionDelaySeconds • 77
retrySeconds • 60
Rollback • 123
roundRobinTypes • 67
routeOnVoucherNumber • 67
routingVoucherNumberLength • 70, 120
Running • 22

S
Scope • v
scps • 81
SCTP • 148
secondary • 48
secondaryConnectionDelaySeconds • 99
Send later flow • 118
serverId • 8, 38
cerPortOverride • 70
ServerHandle • 82
Session • 148
setLastActivationDateStates • 95
SGML • 148
SGSN • 148
shared parameters • 73
sink parameters • 75
slaveLocalSocketDirectory • 71
SLC • 148
SLEE • 149
SLEE.cfg • 30, 41
SMS • 149
SMS statistics subsystem • 27
soLB • 38
source parameters • 77
spoolChunkSize • 75
spoolDirectory • 23, 39, 75
SRF • 149
SSP • 149
Starting the commands • 124
startsAt • 59
Startup • 43, 46, 56, 62, 72, 80, 90, 109, 110, 114, 119, 120, 121
Startup checks • 135
Startup process • 135
State transitions • 23
Statistics • 27
storageInterface • 85
Structure of plugins array • 90
Supported requests • 10, 28
Sync files • 15
Synchronization • 14, 72
Synchronization diagram • 16
Synchronization process • 16
syncIfName • 40
syncMaxDelaySeconds • 97
syncWriter parameters • 102
System Administrator • 149
System Device Configuration • 140
System Overview • 1

T
TCAP • 149
TCP • 149
telco • 149
Telecommunications Provider • 149
throttleLife • 86
Throttling • 12
timeBetweenThrottles • 86
timeout • 45
timerIfName • 38
Tools and Utilities • 123
Tools and Utilities Overview • 123
Tracing parameters • 106
triggerInterface • 86
triggerPlugins • 113
Troubleshooting • 133
Typographical Conventions • vi

U
Update balance parameters • 127
Update bucket parameters • 127
Update process - mfile • 27
Update process diagram • 26
Update wallet parameters • 128
URL • 149
User Interface-Based Configuration Tasks • 30, 41
Username • 84
useTimeFromClient • 95
USSD • 149
UUC Activation • 90

V
Valid interfaces • 80
VLR • 150
Voucher & Wallet Server packages • 139
voucherCache parameters • 100
voucherCreateBufferSize • 105
voucherDeleteBufferSize • 105
voucherReservationPeriodSeconds • 95
voucherUpdateBufferSize • 105
vwarsIfNamePrefix • 41
vwarsTimeout • 69
VWS • 150
VWS associations • 5
VWS Correction Tool • 123
VWS Domains • 3
VWS EDR processing • 25

W
Wallet and bucket events • 6
Wallet and Bucket Expiry • 5, 6, 9, 18
Wallet and bucket expiry processing • 20
Wallet and voucher caches • 9
Wallet life cycle period checks • 11
Wallet life cycle plans • 5
Wallet lifecycle • 5
Wallet management processes • 18
Wallet states • 4
walletCache parameters • 101
walletCreateBufferSize • 105
walletDeleteBufferSize • 106
walletIds • 107
walletLowWaterMark • 99, 100
Wallets • 4
Wallets, Balances and Buckets • 4
Wallets, balances and buckets relationship • 6
walletUpdateBufferSize • 106

X

XML • 150
XmlTcap Parameters • 81
XMS • 147, 150