Oracle® Communications Convergent Charging Controller

Voucher Manager Technical Guide

Release 12.0.6

September 2022

Copyright

Contents

About This Document

Scope

The scope of this document includes all the information required to configure and administer the Voucher Management feature.

Audience

This guide was written primarily for system administrators and persons configuring and administering vouchers in Prepaid Charging. However, sections of the document may be useful to anyone requiring an introduction to vouchers.

Prerequisites

A solid understanding of Unix and a familiarity with IN concepts are an essential prerequisite for safely using the information contained in this technical guide. Attempting to configure or otherwise alter the described system without the appropriate background skills, could cause damage to the system; including temporary or permanent incorrect operation, loss of service, and may render your system beyond recovery.

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:

- *Charging Control Services User's Guide*
- *Advanced Control Services User's Guide*
- *Advanced Control Services Technical Guide*
- *Service Management System Technical Guide*
- *Service Management System User's Guide*
- *Voucher Manager User's Guide*
- *Voucher and Wallet Server Technical Guide*

Document Conventions

Typographical Conventions

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

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

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 Convergent Charging Controller network or service implications of the product.

In this Chapter

This chapter contains the following topics.

Introduction

Vouchers

Vouchers add value to wallet balances.

CCS provides systems for:

- creating Voucher Types and batches
- securing Vouchers (though PIN numbers)
- redeeming Vouchers, and
- automatically deleting archived Vouchers.

CCS uses VWS's voucher system to redeem, query and delete archived vouchers. Redeeming and deleting vouchers are the only tasks performed on the Voucher and Wallet Server. The rest of voucher management is done on the SMS by CCS processes.

Vouchers are sometimes known as scratch cards.

CCS component

Vouchers are part of the Prepaid Charging solution and build on functionality provided by CCS. To fully understand how tasks work, you must also refer to the *CCS Technical Guide*.

Restricted functionality

This functionality is only available if you have purchased the Vouchers licence.

Process descriptions

This table describes the processes involved in voucher creation, redemption and deletion in CCS.

Process	Role	Further information
beServer	Partner of the BeClient. The interface for client requests to the VWS. Supports handler plugins to provide application-specific functionality.	*VWS Technical Guide*
beVWARS	Manages actions against wallets and vouchers. Reads, caches and updates voucher information from E2BE database and writes EDRs. beVWARS plugins perform functions on data in the database.	*beVWARS* (on page 33)
BeClient	Passes voucher redemption messages between control plans on the SLC and the beServer on VWSs.	BeClient
ccsBeAvd	ccsBeAvd runs on a regular basis as a single independent instance on each VWS node. It deletes a range of archived/deleted voucher records from the VWS.	*ccsBeAvd* (on page 34)
ccsBeAvdStartup.sh	Registers ccsBeAvd as a replication VWS node and starts the ccsBeAvd process.	
ccsVWARSVoucherHandler	ccsVWARSVoucherHandler is a beVWARS message handler which handles reserving a voucher and redeeming a voucher.	*ccsVWARSVoucherHandler* (on page 39)
ccsVWARSWalletHandler	ccsVWARSWalletHandler is a beVWARS message handler which performs the wallet changes specified by a successful voucher redemption.	ccsVWARSWalletHandler
ccsVoucher_CCS1 and ccsVoucher_CCS3	These ccsVoucher tools generate batches of vouchers into the SMF database. They also perform some batch updates.	
ccsVoucherStartup.sh	New Voucher Batch screen collects parameters, and runs ccsVoucher_CCS1 or ccsVoucher_CCS3 through ccsVoucherStartup.sh.	*ccsVoucherStartup.sh* (on page 17)
Security libraries	ccsVoucher uses security libraries to encrypt voucher batches. ccsVWARSVoucherHandler uses security libraries to decrypt PIN details when redeeming vouchers.	Security libraries

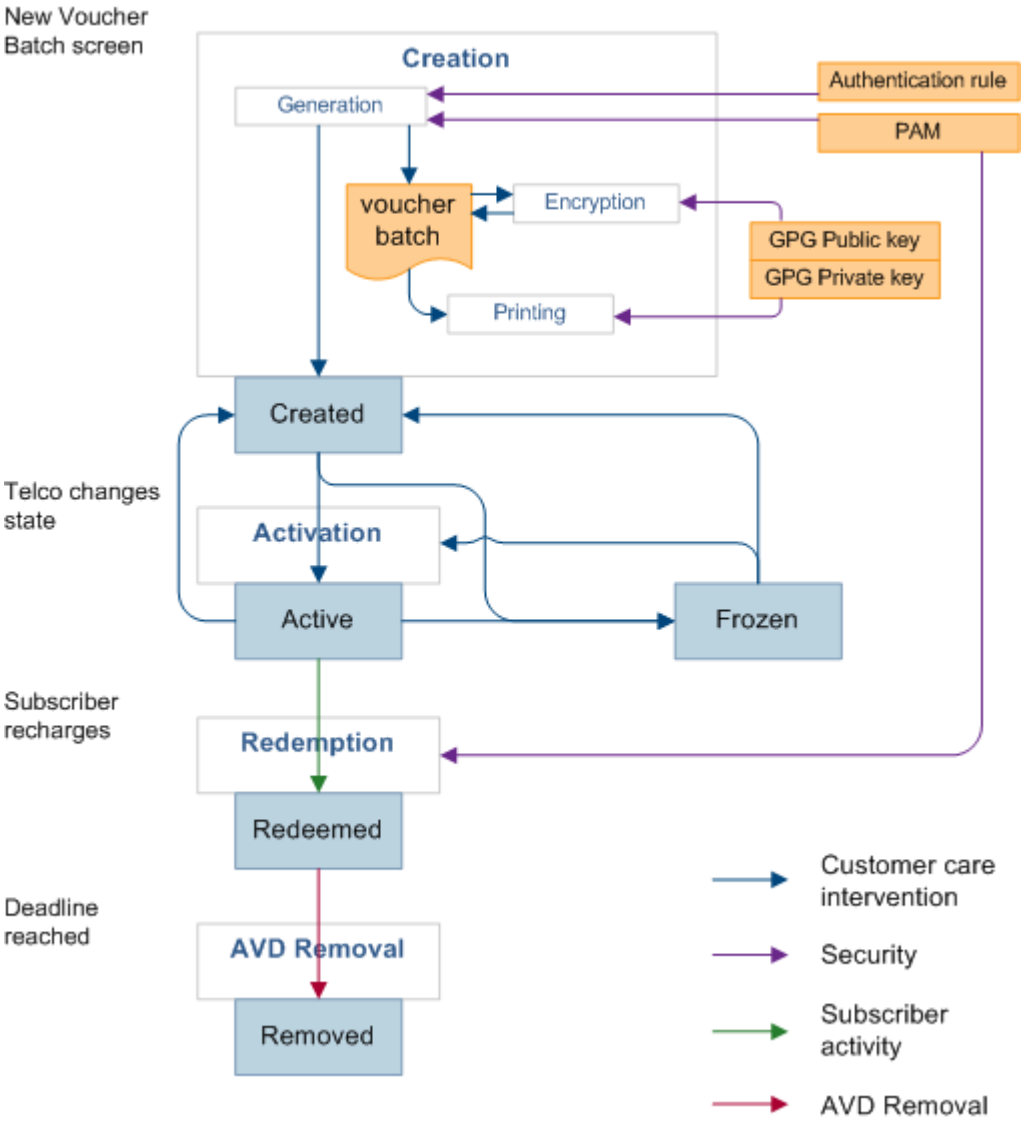
Process	Role	Further information
CCS Macro Nodes	CCS includes a specific set of voucher nodes which enable subscribers to use self-care IVR systems to redeem vouchers.	*CCS Feature Node User's Guide*
CCS Java Administration Screens	The CCS Admin screens enable users to: • configure and generate new batches of vouchers • perform one-off voucher redemptions. **Note:** The way EDRs look in Prepaid Charging's subscriber management screens is directly affected by a configuration file. See cdrDetailsConfig.conf for CCS.	*CCS User's Guide*

Voucher Lifecycle

Voucher life cycle

This diagram shows the life cycle of a voucher.

Note: Voucher batches have a slightly different life cycle.



Voucher batches and CCS

Voucher batches are created either by:

1. running the ccsVoucher_CCS3 script directly, or
2. using the Create Voucher Batch screen via the Vouchers menu from CCS.

The bulk of the voucher creation work is done by the command-line tool, ccsVoucher_CCS3. The Create Voucher Batch screen and ccsVoucherStartup.sh script provide a simple graphical wrapper for the ccsVoucher process. The screen performs a consistency check before the voucher-creation request is sent to the SMS host through ccsSmsORB and smsReportsDaemon.

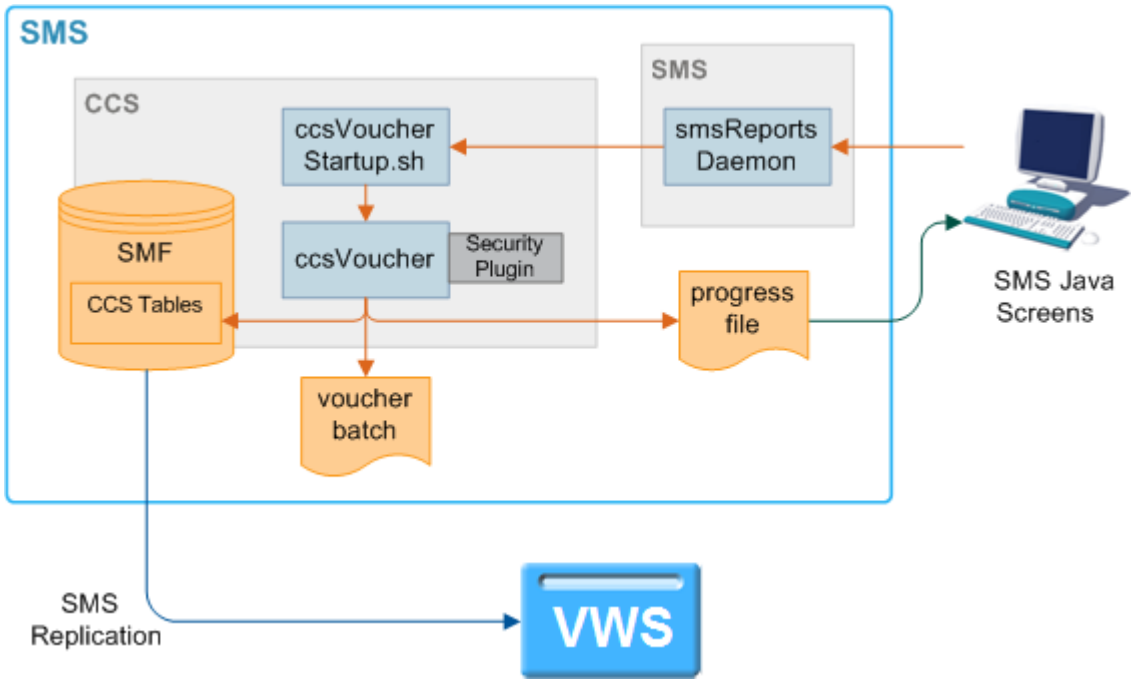
If you are using the Create Voucher Batch screen, after the ccsVoucher_CCS3 job starts at the background, the process may fail, even though the screen indicates that the job has started successfully.

Generating vouchers

ccsVoucherStartup.sh (on page 17) runs the *ccsVoucher_CCS3* binary in create mode for CB10 HRN SHA and CB10 HRN AES encryption. The voucher batch is created on the SMS and transferred from there to Voucher and Wallet Server.

Generating vouchers diagram

This diagram shows the processes involved in generating a batch of vouchers.



Vouchers and VWSs

Once a batch of vouchers have been generated on the SMS, the full batch is divided up into groups and each group is replicated completely to one VWS pair only. They are only replicated to VWSs which have been configured to be in a Domain which supports vouchers. Each voucher is only replicated to a single VWS pair. The voucher records on each VWS include a record of which BE the voucher should be on.

No voucher information is replicated to the SLCs. BeClient does not maintain Voucher location information. When a voucher redeem is triggered on the SLC, the BeClient process broadcasts to all VWS pairs asking for the VWS pair with that voucher on it. The VWS pair that owns that voucher then replies to the request indicating it holds the relevant voucher data and makes a voucher reservation.

Changing voucher states

ccsVoucher binaries (*ccsVoucher_CCS1* or *ccsVoucher_CCS3*) has two options which change voucher states:

- activate, and
- state.

If ccsVoucher binary is run with the activate option, it will attempt to change the state of a batch of vouchers to 'activate'.

If ccsVoucher binary is run with the state option, it will attempt to change the state of a range of vouchers within a batch to a specified state.

The maximum number of vouchers for one job is 999999999. The operation will commit in batches of 100, and pause for the specified time between commits.

Vouchers that have already been redeemed may not have their state changed.

Triggering a voucher redemption

CCS obtains Voucher information for recharges from subscribers using subscriber interaction using these systems:

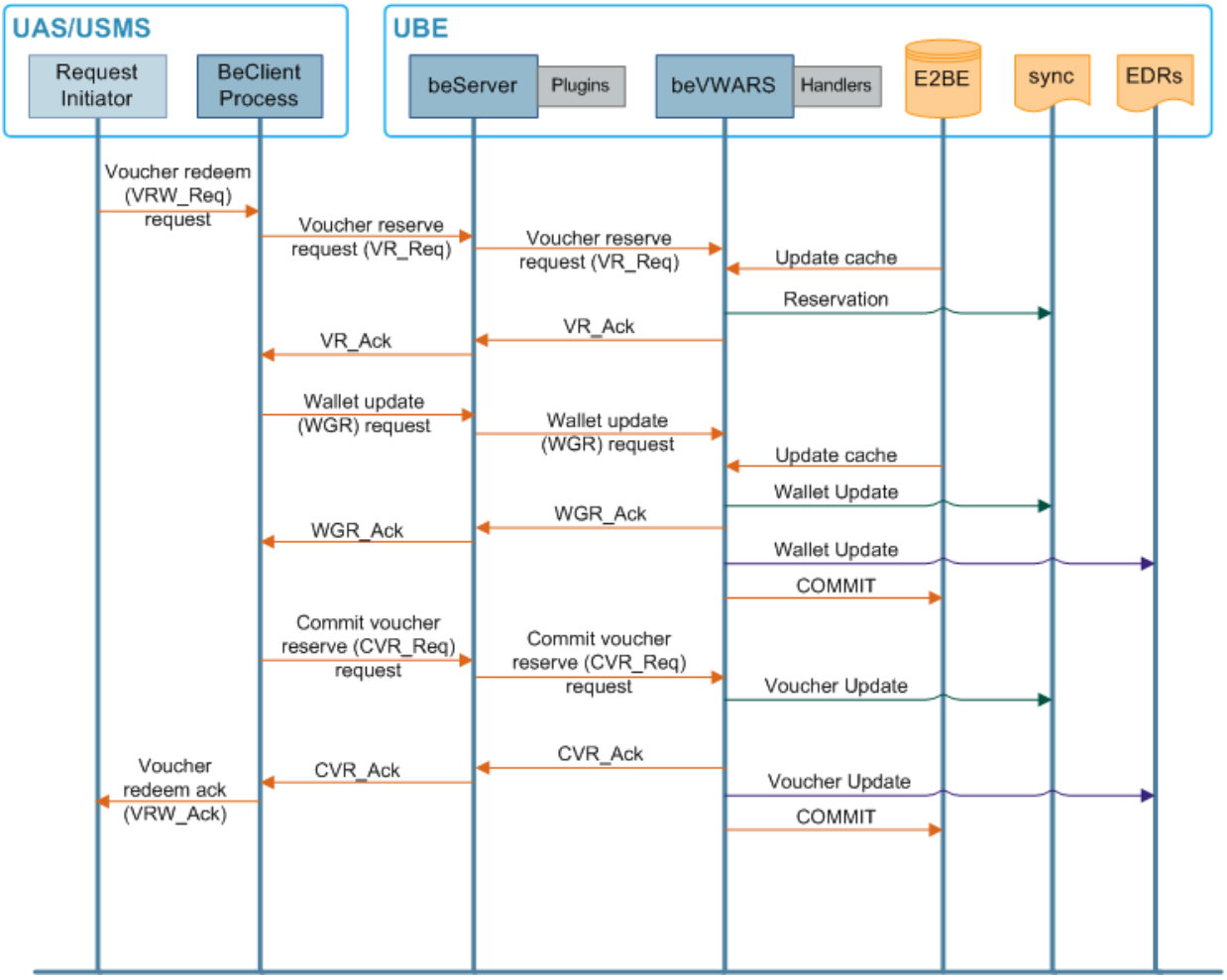
- IVR feature nodes in a control plan
  - subscribers can recharge their account by providing Voucher details through IVR (using the CCS voucher nodes)
- interaction with a customer services representative (who uses the Voucher Management screen)
- (if MM is installed) Short Messages sent from the subscriber's handset, and
- (if USSD GW is installed) menus and fast access.

The validation of the Voucher is done by the chargeable service (usually CCS) through beVWARS plugins on the VWS.

Voucher redemption message flows

This diagram shows the message flows for a voucher redeem.

Note: This message flow is valid where both the voucher domain and the wallet domain are VWS. If the domains are different, the message flow will return back to the Request Initiator each time, so it can pass the next request to a different client. For more information about this type of voucher redemption, see the Technical Guide for the non-VWS VWS client involved in the processing.



Voucher redeem process

This table describes the stages involved in redeeming a voucher using CCS plugins. For more information about the VWS actions in this process (including EDR writing), see *VWS Technical Guide*.

Note: This documentation describes a voucher redemption where both the voucher and wallet domains are VWS. If the wallet domain is different from the voucher domain, the messages will return back to the process which triggered the redemption. The initiating process will then pass a wallet recharge request to the VWS client which is configured to handle the wallet's domain. For more information about this type of voucher redemption, see the Technical Guide for the non-VWS VWS client involved in the processing.

Stage	Description
1	Voucher redemption is triggered using any of the methods described in *Triggering a voucher redemption* (on page 6). The information is collected from the user and passed to the relevant BeClient process. • For control plans, the request is passed to BeClient on the SLC. • For Voucher Management screen redemptions, the request is passed to ccsBeOrb on the SMS.

Stage	Description
2	Voucher requests are sent with a BE id of 0, so the libclientBcast plugin handles broadcasting the Voucher Reserve (VR_Req) request to all VWS Voucher and Wallet Servers.
3	beServer processes on the Voucher and Wallet Servers pass the request to the a beVWARS process.
4	The beVWARS process checks for VR message handlers. CCS provides *ccsVWARSVoucherHandler* (on page 39) for VR messages, so beVWARS passes the message to that handler. ccsVWARSVoucherHandler checks whether: • this VWS holds the details for the requested voucher, and • whether the voucher can be redeemed (if the voucher's PIN was encrypted, ccsVWARSVoucherHandler will use the Security libraries to check the PIN). If the voucher is local, and can be redeemed, ccsVWARSVoucherHandler reserves the voucher and passes the voucher details back to the BeClient via beServer. The beVWARS processes on VWSs which do not hold the voucher's details return a VR_Nack. If none of the available VWSs hold the voucher details, BeClient will send a create EDR request (CCDR_Req) to a VWS so the beVWARSCCDRHandler will log an EDR recording the failed voucher redemption. Go to stage 9.
5	The BeClient process sends a Wallet General Recharge (WGR_Req) request back to the beVWARS via the beServer. **Note:** This request may be to a different VWS from the one which has reserved the voucher.
6	beVWARS checks for WGR message handlers. CCS provides ccsVWARSWalletHandler for WGR messages, so beVWARS passes the message to that handler. ccsVWARSWalletHandler updates buckets/balances specified in the request by the amount specified in the request. beVWARS then passes a WGR_Ack back to the BeClient via beServer. If the wallet recharge fails, beVWARS return a WGR_Nack.
7	If the wallet recharge succeeds, BeClient sends a Commit Voucher Reservation (CVR_Req) request back to beVWARS via beServer. If the wallet recharge fails, BeClient sends a Revoke Voucher Reservation (RVR_Req) request to beVWARS via beServer.
8	beVWARS checks for CVR or RVR message handlers. CCS provides *ccsVWARSVoucherHandler* (on page 39) for both types of messages, so beVWARS passes the message to that handler. If the message was a CVR, ccsVWARSVoucherHandler updates the voucher to redeemed and returns a CVR_Ack to BeClient via beServer. If the message was a RVR, ccsVWARSVoucherHandler clears the reservation set in stage 4 and returns a RVR_Ack to BeClient via beServer.
9	If the recharge was successful, BeClient returns a VRW_Ack to the requesting process. If the recharge failed, BeClient returns a VRW_Nack.

Note: If there are any beServer message handlers configured for the messages passed in this process, beServer will pass the request its handlers before passing them to the beVWARS. These handlers may generate other messages to be passed to the beVWARS, and the handlers on the beVWARS will handle them before responding back to the beServer handlers. CCS does not provide beServer message handlers for any of the messages described in this process.

Automatic voucher deletion

Vouchers that have been redeemed will be archived on a weekly basis. The archived vouchers will be automatically deleted from the VWS after a configurable number of weeks has elapsed.

For more information on automatic voucher deletion see the *Voucher Manager User's Guide*.

Security

Authenticating modules

To provide security over account and voucher generation, CCS contains authentication modules.

These modules contain information uniquely related to the account or voucher number, which is not stored (directly) in the database, but which must be supplied in order to make use of the account or voucher.

Each module has a pair of functions.

1. The first function (the hash generation function) is called at subscriber account- or voucher-generation time.
2. The second (the hash validation function) is called every time a subscriber account- or voucher number is presented to the system during call processing.

Note: Once a batch is created, the authentication module associated with that batch may not be changed.

Voucher PINs

Without PIN validation, subscribers may attempt to guess valid voucher numbers. PINs are stored in CCS_VOUCHER_REFERENCE table.

PIN numbers for Vouchers are implemented through security plugins. These plugins are used by:

- *ccsVoucher_CCS1* (on page 38) and *ccsVoucher_CCS3* (on page 38) to generate voucher PINs, and
- beVWARS *ccsVWARSVoucherHandler* (on page 39) plugin to check PIN numbers for validity.

Tip: The plugin used to generate the vouchers is also used for validation.

Modules and security plugins

This table describes when security plugins are used and which process they are used by.

Process	Use
ccsAccount	Used to generate subscriber account or calling card PINs (which are used to secure self-management systems).
ccsVoucherStartup.sh (on page 17)	Used to generate voucher PINs (that is, a string of digits to be printed on the voucher or scratch card.
beVWARS *ccsVWARSVoucherHandler* (on page 39) plugin	Used to check PIN numbers for validity (for example, to validate a string of digits entered by the user indicating a Subscriber account to use or a voucher to redeem).

For more information about the ccsAccount tool, see *CCS Technical Guide*.

Security libraries

Voucher management uses security libraries to provide flexibility in how the PINs are generated by *ccsVoucher_CCS3* (on page 38). This table describes the function of each security library.

Library	Description
ccsLegacyPIN (on page 38)	Provides the DES authentication module (DES crypt()ed n-digit PINs) for account/voucher authentication. Used by subscriber account and voucher subsystems. The plug-in library is not applicable to new voucher batches. **Note:** The output file is sent directly to the third-party tool gpg, so the resulting printer file is encrypted. The printer file is never created on the SMS in an unencrypted format.
ccsCB10HRNSHA	Provides the CB10 HRN SHA256 and CB10 HRN SHA512 authentication modules for voucher security.
ccsCB10HRNAES	Provides the CB10 HRN AES256 authentication module for voucher security.

The authentication module is selected in the New Voucher Batch screen. For information about this screen, see *Voucher Manager User's Guide*.

Tip: Vouchers are validated using the same plugin as they were generated with.

DES Encryption

DES Encryption supports separate voucher number and voucher PIN generation.

The generated voucher numbers will be determined using the start and end range values specified for the voucher batch, while the voucher PINs will be randomly generated.

The length of the voucher number and the voucher PIN will depend on the configuration specified for the DES encryption rule being used.

Where DES Encryption is used, gpg is used to encrypt the exported voucher batch file.

Public and private key encryption

Public and private key encryption (also known as asymmetric encryption) involves a pair of keys:

1 a public key which is used encrypt the file, and

2 a private key which is used to decrypt the file.

Both keys are generated by the holder of the private key. The public key is made available to others who want to send encrypted files to the private key holder. In this case, the print shop will generate the public and private keys and provide the public key to the operator.

For more information about:

- generating keys, see Managing Public/Private Key Pairs.
- decrypting files, see Decrypting Files.

More information about public and private key encryption is widely available in publications and on the Internet.

GPG keys

GPG Public keys are used to increase security when creating subscriber account and voucher batch export files for printing.

To use GPG public keys, you must use the Voucher Management screen to:

- Import new GPG public keys
- Verify the imported keys.

Note: You cannot use a key until you verify it.

When a GPG Public Key is imported, it is added to the SMF database by smf_oper. When verified, they are marked as verified. These keys are then available when creating a voucher or account batch. You cannot remove public keys from the database or from the GPG key-ring store on the SMS.

When a voucher batch is created a required key or UID will be supplied. The UID is used to determine which GnuPG key to use within the keyring to encrypt the export file. The key UID is a hexadecimal number up to 10 digits in length.

For more information about the Voucher Management screen, see *Voucher Manager User's Guide*.

CB-10 HRN Creation

The CB10 HRN creation provided by the CB10 HRN authentication module generates voucher numbers using the:

- CB-10 HRN private keys (K1, K2 and K3) for the service provider
- S, R1, R2 and R3 security parameters defined for the authentication rule

The CB-10 HRN creation algorithm does not support voucher PINs and therefore the PIN length is always set to 0.

A unique set of CB-10 HRN private keys (K1, K2 and K3) is required for each service provider. These keys are generated in one of the following ways:

- Generated automatically when a new service provider is created
- Generated or regenerated for service providers who existed before the CB-10 HRN authentication was activated using the **Generate** button for one of the CB10 HRN SHA or AES PAM modules

Note: If a voucher batch is already created for a service provider using the CB-10 creation algorithm then you cannot:

- Regenerate the K1, K2, and K3 private keys for the selected service provider
- Edit the associated authentication rule

CB-10 HRN SHA Hashing

CB-10 HRN private keys are created when:

- a new service provider is created
- a **Generate** button is clicked for a CB10 or SDK PAM Authentication module and the service provider does not have any CB-10 vouchers created yet

Clicking a **Generate** button also generates hash/encryption keys for the specific PAM Authentication module that are used for hashing/encrypting the CB10 or SDK-created HRN. **Generate** buttons are disabled if the service provider already has a voucher batch created using the specific PAM Authentication module.

For example, after a voucher batch is created for a service provider using an authentication rule specifying a PAM of CB10 HRN SHA256, you cannot change or update the hash key for CB10 HRN SHA256.

PAMs that use SHA hashing can specify the number of hash iterations to apply, the default is 1 iteration.

Decryption to retrieve the HRN is not supported for SHA-based PAMs as the SHA operation is not reversible.

Chapter 2

Configuration

Overview

Introduction

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

In this chapter

This chapter contains the following topics.

eserv.config Configuration

Introduction

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

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

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

Configuration File Format

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

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

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

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

or

```
{ name="route6"
    id = 3
    prefixes = [
        "00000148"
        "0000473"
    ]
```

```
    }
    { name="route7"
        id = 4
        prefixes = [
            "000001049"
        ]
    }
```

or

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

eserv.config Files Delivered

Most applications come with an example **eserv.config** configuration in a file called **eserv.config.example** in the root of the application directory, 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 causes file errors when the application tries to read the configuration file.

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

Loading eserv.config Changes

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

Configuring VWS processes for CCS

VWS processes used by CCS

beVWARS on the VWS must be configured to include the CCS beVWARS plugins and message handlers for voucher functionality.

For more information about configuring ccsVWARSVoucherHandler, see:

- *beVWARS* (on page 33), and
- *ccsVWARSVoucherHandler* (on page 39).

Message handlers and event plug-ins

Message handlers provide functionality which is specifically related to messages passed between BeClient and the VWS. Plug-ins are designed to handle specific events such as a balance expiry date being passed.

Message handlers

CCS installs a number of message handlers and plugins into the VWS for handling the CCS-specific messages and functionality. For handling voucher messages, CCS provides *ccsVWARSVoucherHandler* (on page 39).

Other handlers are described in *CCS Technical Guide*.

Chapter 3

Tools and Utilities

Overview

Introduction

This chapter provides a description of the operational programs or executables which are used to administer CCS. All of these processes are performed when needed.

Executables are located in the `/IN/service_packages/CCS/bin` directory.

Some executables have accompanying scripts that run the executables after performing certain cleanup functions. All scripts should be located in the same directory as the executable.

Note: Most processes can be re-started using the UNIX kill command.

Using SLP Trace log files

Processes started by the inittab and cronjobs produce logfiles that are stored in the tmp folder of each service directory, that is:

```
/IN/service_packages/CCS/tmp/
```

Other CCS tools

The other CCS tools are documented in the *CCS Technical Guide*.

In this chapter

This chapter contains the following topics.

ccsVoucherStartup.sh

Purpose

Depending on the option used, the ccsVoucherStartup.sh:

- generates batches of vouchers from data entered on the command line
- changes the state of vouchers in a specified range
- activates a batch of vouchers
- cancels voucher batches, and
- cleans up expired vouchers.

For more information about these processes, see *Vouchers* (on page 1).

All voucher management and administration is performed on the SMS.

ccsVoucherStartup.sh runs one of the ccsVoucher binaries, passing on any relevant configuration as command line parameters. The two ccsVoucher binaries are:

1 *ccsVoucher_CCS1* (on page 38), and

2 *ccsVoucher_CCS3* (on page 38).

Location

This binary is located on the SMS node.

Startup

Follow these steps to run the ccsVoucher tool.

Step	Action
1	Login to the SMS machine on which your CCS application is installed as ccs_oper.
2	Navigate to the directory in which ccsVoucher is located. In a standard installation, this will be: `/IN/service_packages/CCS/bin`
3	Enter the program name: `ccsVoucher option parameters` **Where:** ccsVoucher is one of the binaries - ccsVoucher_CCS1 or ccsVoucher_CCS3 *Options* (on page 18) and *Common command line parameters* (on page 18) are defined in the following tables. **Note:** The option determines the action to be performed. The subsequent parameters depend on the option selected.

Options

The following options define the task ccsVoucher will perform.

`<option>`

Syntax:	*option*	
Description:	*option* specifies the task ccsVoucher will perform.	
Type:	String	
Optionality:	Required	
Allowed:	activate	Change the state of a batch of vouchers to 'activated'.
	cancel	Mark a batch as cancelled in the SMF database.
	cleanup	Mark batches unusable if beyond their pre-use expiry.
	create	Generate a new batch of vouchers.
	state	Change the state of a range of vouchers within a batch to a specified state.
Default:	None	
Notes:	The parameters that may be used depend on the specified option.	
Example:	`ccsVoucher create parameters_list`	

Common command line parameters

The following command line switches and parameters are common to both ccsVoucher_CCS1 and ccsVoucher_CCS3.

-v provider

Syntax: -v *provider*

Description: The name of the service provider.

Type: String

Optionality: Optional

Allowed:

Default: null

Notes:

Example:

-r start[/end]

Syntax: -r *start*[/*end*]

Description: *start* is the lowest serial number in the range of vouchers

end is the highest serial number in the range of vouchers

Type: Integer

Optionality:

- Start voucher number is required
- End voucher is required for state option
- End voucher number is optional for create option, but if it is not specified, the size of the batch (-*s size* (on page 19)) must be specified

Allowed: End voucher must be a higher number than start voucher.

Default: None

Notes: Valid for create and state options

Example: -r 1234567440/1234567489

-s size

Syntax: -s *size*

Description: *size* is the total number of vouchers to be produced in the batch.

When used with -O (On-demand mode), this is the batch size for On-demand vouchers.

Type: Integer

Optionality: Required (must be set if -r or -R do not have an *end* option set)

Required, when used with -O (On-demand) option.

Allowed: Positive integer

Default: None

Notes: Only valid for the generate option

This is not necessarily required because ccsVoucher is able to determine the size of the batch from the start and end ranges of the batch for both the -r and -R arguments (if <*end*> is supplied).

Example: -s 1000

-s state

Syntax: -s *state*

Description: The state the vouchers will be changed to

Type: String

Optionality:	Required
Allowed:	C Created A Activated F Frozen
Default:	none
Notes:	Valid for state option only
Example:	`-s A`

`-B batch_code`

Syntax:	`-B` *`batch_code`*
Description:	*`batch_code`* is the code of the batch to perform the operation on.
Type:	Integer
Optionality:	
Allowed:	
Default:	None
Notes:	This cannot be used with `-b` option Valid for all options except cleanup.
Example:	`-B 362`

`-t type`

Syntax:	`-t` *`type`*
Description:	The name of the voucher type.
Type:	String
Optionality:	Required (default raises error)
Allowed:	This must be the same as the name defined in the NAME field in the CCS_VOUCHER_TYPE table.
Default:	Null
Notes:	Used in create and cancel modes.
Example:	

`-m pam`

Syntax:	`-m "`*`pam`*`"`
Description:	The name of the authentication module to use when generating vouchers.
Type:	String
Optionality:	Required

Allowed:	AltAuthMod	Use ccsVoucher_CCS1 with -B.
		Note: This option is deprecated.
	DES (no key set)	Use ccsVoucher_CCS3 with no special parameters.
	DES (and a *keyUid* (on page 24) set)	Use ccsVoucher_CCS3 with -B, -F, -R, -S, -n, and -o.
	DES Encryption	Use ccsVoucher_CCS3 with no special parameters.
		Note: This option is deprecated.
	CB10 HRN, CB10 HRN SHA256 or CB10 HRN SHA512 (no key set)	Use ccsVoucher_CCS3 with -B, -F, -R and -n.
	CB10 HRN (and a *keyUid* (on page 24) set)	Use ccsVoucher_CCS3 with -B, -F, -R, -n and -o.
Default:	No default	
Notes:	This parameter is set by the PAM field in the New Authentication Rule screen.	
Example:	`-m "DES"` `-m "CB10 HRN SHA256"`	

-M rule

Syntax:	`-M "`*`rule`*`"`
Description:	The name of the authentication rule to use when generating a batch of vouchers.
Type:	String
Optionality:	Required
Allowed:	Must match an Authentication Rule name in the Security tab on the Service Management screen.
Default:	No default
Notes:	The authentication rule associated with a voucher batch determines which encryption algorithm to use when generating the voucher numbers in the batch.
Examples:	`-M "CB10 S=10"` `-M "DES (CCS3) NL=10 PL=04"`

-c str

Syntax:	`-c `*`str`*
Description:	The voucher context data.
Type:	String
Optionality:	Optional (default used if not set)
Allowed:	
Default:	Null
Notes:	Only used with create mode
Example:	

`-e be`

Syntax: `-e be`
Description: The Voucher and Wallet Server name.
Type: String
Optionality: Required (default raises an error)
Allowed:
Default: null
Notes: Used with create and cancel modes
Example:

`-f filename`

Syntax: `-f "file"`
Description: For the create option, *file* is the output voucher file name.
For the state option, *file* is the file to which failed changes are written.
Type: String
Optionality:
Allowed:
Default: Null
Notes: The file will be written to the export directory for both options.
Examples:
```
-f "Batch362Generation.txt"
-f "Batch362ChangeErrors.txt"
```

`-p pause`

Syntax: `-p secs`
Description: The number of seconds to pause between the generation or change of a sub-set of vouchers.
Type: Integer
Optionality:
Allowed:
Default: 0 - For create option.
Notes: Valid for create and state options.
Example: `-p 6`

`-D str`

Syntax: `-D "str"`
Description: EDR description.
Type: String
Optionality: Optional (default used if not set)
Allowed:
Default: Null
Notes: Only used with create mode.
This parameter is populated by the CDR Description field in the New Voucher Batch screen.
Example: `-D "Batch-New_CB10_0003"`

-P file

Syntax: -P *path_file*

Description: The file to send the results to (success or failure) when ccsVoucher is executed through the UI.

Type: String

Optionality: Required (default raises an error)

Allowed:

Default: null

Notes: Used with all states

Example:

-i state

Syntax: -i *state*

Description: The state vouchers created in this batch will have when they are first generated.

Type: String

Optionality: Required when used for generating new vouchers. When used for changing the state of a range of vouchers, -i is optional, (default used if not set).

Allowed:

C	Generate vouchers with state = Created
A	Generate vouchers with state = Activated
F	Generate vouchers with state = Frozen

Default: When used to change the state of a range of vouchers, the default is A.

Otherwise there is no default.

Notes:

Example: -i C

-b batch id

Syntax: -b *batch_id*

Description: <batch id> is the batch id to generate or the batch to perform the operation on. Corresponds to the voucher batch's database ID.

Type: Integer

Optionality: Required

Allowed:

Default: None

Notes: This cannot be used with the -B option.

Valid for all other options except cleanup.

Example: -b 362

-u usr/pwd

Syntax: -u *usr*[/*pwd*]

Description: *usr* is the username of the Oracle account to use to log into the SMF database.

pwd is the password of the Oracle account to use to log into the SMF database.

Type:

Optionality: Optional (default used if not set)

Allowed:

Default:	/
Notes:	Valid for all options.
Example:	`-u smf/smf`

`keyUid`

Syntax:	*keyUid*
Description:	The id of the gpg key to use to encrypt the output file.
Type:	String
Optionality:	Optional (file not encrypted if not set)
Allowed:	
Default:	null
Notes:	Not available with the AltAuthMod PAM.
Example:	

`-d`

Syntax:	`-d`
Description:	Perform a dry run. Checks parameters only, does not create any vouchers.
Type:	Boolean
Optionality:	Optional (default used if not set)
Allowed:	
Default:	Not set
Notes:	
Example:	

Create example

To produce a set of vouchers, both serial number and a voucher number must be produced for each voucher. These numbers are sequential starting from some number supplied by the user.

All command line parameters are optional, but you'll be prompted for the following if batch ID is not provided:

- Voucher and Wallet Server
- service provider, and
- batch details.

To produce ten vouchers with the serial numbers starting at 100 and the voucher numbers starting at 200, the switches and parameters could be either of:

```
ccsVoucher create -m "DES" -r 100 -R 200 -s 10
ccsVoucher create -m "DES" -r 100/109 -R 200/209
```

Monitoring voucher generation

You can check the ongoing status of the background ccsVoucher job by reading the ccsSmsORB output log. Any error messages from the actual voucher creation process will appear in there. For information about locating the ccsSmsORB log, see ccsBeOrb Details.

If the job is successful, a file named as specified in the voucher batch creation screen, with '.lst.print' appended to the filename, will appear in the following directory with a non-zero file size:

/IN/service_packages/CCS/voucher/export/

If the job has failed, then either this file will not be created, or it will contain no data (that is, be zero bytes long), or no usable data (that is, headers only, no voucher data).

State example

To activate 40 vouchers with serial numbers starting at 100, the options and parameters would be:

```
ccsVoucher state -r 1234567440/1234567489 -p 6 -f Batch362ChangeErrors.txt
```

Note: You don't need to set -s A, as this is the default.

CCS3 command line parameters

The following command line switches and parameters are specific to *ccsVoucher_CCS3* (on page 38). They can be used with the CCS3 Encryption and CCS1 Compatable methods of generating vouchers.

`-F batch_name`

Syntax:	`-F str`
Description:	The name of the context file.
Type:	String
Optionality:	
Allowed:	String must be 50 characters or less.
Default:	
Notes:	This cannot be used with `-B`.
Example:	

`-n`

Syntax:	`-n`
Description:	Allow overlapping voucher ranges.
Type:	
Optionality:	Optional (default used if not set)
Allowed:	
Default:	Not set. Don't allow overlapping voucher ranges.
Notes:	The `-R` option must also be specified.
Example:	

`-o`

Syntax:	`-o`
Description:	Do not send the vouchers to the export file. Use standard out instead.
Type:	
Optionality:	
Allowed:	
Default:	
Notes:	
Example:	

`-O`

Syntax:	`-O`
Description:	Create On-Demand Voucher. To be used in create mode only.
Type:	
Optionality:	Optional (default used if not set)
Allowed:	
Default:	Not set. Don't create 'On-Demand Voucher' when not specified.
Notes:	
Example:	

`-R`

Syntax:	`-R`
Description:	Create random voucher number.
Type:	
Optionality:	
Allowed:	
Default:	
Notes:	
Example:	

`-S`

Syntax:	`-S`
Description:	Create random salt for CCS1 Legacy or CCS3 Encryption.
Type:	
Optionality:	
Allowed:	
Default:	
Notes:	The Unix 'crypt' function uses a 2-digit alphanumeric string (SALT) which is used as part of the encryption. By default, the SALT is fixed. To increase security you can also randomly generate the SALT for each voucher created. This provides a non-uniform private/secret key.
Example:	

eserv.config parameters

ccsVoucherStartup.sh accepts the following global configuration parameters in **eserv.config**.

Note: SMS only.

`disableConcurrencyLock`

Syntax:	`disableConcurrencyLock = true|false`
Description:	Determines whether or not multiple instances of ccsVoucher are allowed to run concurrently.
Type:	Boolean
Optionality:	Optional (default used if not set)

Allowed:	true	Do not perform the locking file checking of the file specified by *lockFileName* (on page 27).
	false	Use lock file to ensure only one ccsVoucher process is running at a time.
Default:	false	
Notes:	This will not disable the checking done in the wrapper script.	
Example:	`disableConcurrencyLock = false`	

displayVoucherValue

Syntax:	`displayVoucherValue = <true|false>`
Description:	Whether or not to include voucher values in batch report
Type:	Boolean
Optionality:	Optional (default used if not set)
Allowed:	
Default:	false
Notes:	
Example:	`displayVoucherValues = false`

ignoreRandomGenerationFlags

Syntax:	`ignoreRandomGenerationFlags = true|false`
Description:	Ignore the command line parameters used for random voucher number generation (-R and -n).
Type:	Boolean
Optionality:	Optional (default used if not set)
Allowed:	
Default:	false
Notes:	
Example:	`ignoreRandomGenerationFlags = false`

lockFileName

Syntax:	`lockFileName = "`*`path_file`*`"`
Description:	The full path and filename for the lock file.
Type:	String
Optionality:	Optional (default used if not set).
Allowed:	
Default:	**`/IN/service_packages/CCS/logs/.ccsVoucher-lock`**
Notes:	Prevents multiple instances of ccsVoucher running concurrently when set.
Example:	

suppressHeaders

Syntax:	`suppressHeaders = true|false`
Description:	Whether to suppress default header fields in the voucher batch file.
Type:	Boolean
Optionality:	Optional (default used if not set)

Allowed:	true false
Default:	false
Notes:	
Example:	

`voucherFileHeaderPlugins`

Syntax:	`voucherFileHeaderPlugins = [`
	`  "path/lib"`
	`  ...`
	`]`

Description:	Full path to the file writer plugin, if any. If set, then this plugin will be used to format the voucher batch file.
Type:	Array
Optionality:	Optional (default plugin used if not set)
Allowed:	
Default:	
Notes:	
Example:	

`voucherFileWriterPlugin`

Syntax:	`voucherFileWriterPlugin = path/lib`
Description:	Full path to the file writer plugin, if any. If set, then this plugin will be used to format the voucher batch file.
Type:	String
Optionality:	Optional (default writer used if not set)
Allowed:	
Default:	
Notes:	
Example:	

Example eserv.config parameters

Here is an example of the global configuration options parameters for the ccsVoucher tool.

```
ccsVoucher = {
    suppressHeaders = false
    displayVoucherValue = false
    ignoreRandomGenerationFlags = false
    voucherFileWriterPlugin = "IN/service_packages/CCS/lib/yourFilePlugin.so"
    voucherFileHeaderPlugins = [
        "IN/service_packages/CCS/lib/yourHeaderPlugin1.so"
        "IN/service_packages/CCS/lib/yourHeaderPlugin2.so"
    ]
    lockFileName = "/IN/service_packages/CCS/logs/.ccsVoucher-lock"
    disableConcurrencyLock = false
}
```

Failure

If ccsVoucher fails, either:

- no voucher batches will be produced, or
- a partial voucher batch will be created.

Output

ccsVoucher produces:

- Database inserts for the SMF database:
  - voucher number
- Flat file:
  - voucher number
  - voucher PIN

It also logs errors to:

```
ccsVoucher.log
```

Exported voucher batch files

Voucher batch file format is controlled by the security library, and the voucher writer plugin used to generate the batch. Which libraries and plugins are used is defined by the Authentication Module (PAM) and the Authentication Rule specified in the New Voucher Batch screen.

Header fields are in the format "<Key field name>=<value>". Key field names always start with an alphabetic character. This makes it easy to distinguish them from voucher records (which always start with a number).

The following header fields are used in the voucher batch file header, (although downstream processors should detect any "<Key field name>=<value>" lines).

Header field	Description
BilingEngineName=<str>	The name of the Voucher and Wallet Server where the voucher resides.
VoucherTypeName=<str>	The name of the voucher type as created on the Convergent Charging Controller platform. The voucher type contains the following information: • Pre-use expiry period (number of days and hours that this voucher is valid in a pre-use state) • Wallet expiry period (change the current wallet expiry date by this many days and hours) • Voucher number length • Voucher PIN length • A list of all the balance types, associated values and balance expiry date modifications which will be changed/updated when this voucher is redeemed **Note:** It will be up to the operator to provide the details of the voucher type described here to the print shop so that any specific voucher details can be printed on the final vouchers.
AuthRuleName=<str>	The name of the authentication rule which was used for creating the voucher number and PIN.
AuthModName=<str>	The name of the pluggable authentication module (PAM) (Convergent Charging Controller specific) used for creating the voucher PIN.
VoucherBatchBatch=<str>	A two character identifier (non unique) for this voucher batch.

Header field	Description
VoucherBatchID=<int>	The system generated ID for this voucher batch.
OriginalCount=<int>	The number of vouchers created in this batch.
StartOfRange=<int>	Beginning of the range of voucher numbers.
EndOfRange=<int>	End of the range of voucher numbers.

A line consisting of a single equal sign (=) terminates the header lines. All subsequent lines are voucher detail records.

CCS3 DES voucher batch example

This text shows an example export voucher batch file generated by ccsVoucher_CCS3 using the DES encryption library (and a bespoke voucher file writer plugin to format the non-header details), but no GnuPG key.

```
#
# Voucher file for batch 83
# Generated by ccsVoucher at Tue Nov 11 12:55:27 2008
# (key=value or
voucherserialnumber,vouchernumber,vouchersecret,vouchercontext,voucherprivate_secret
)
#
BillingEngineName=PCDEV
VoucherTypeName=DES
AuthRuleName= DES (VL=10 VP=4)
AuthModName=DES
VoucherBatchBatch=
VoucherBatchID=83
OriginalCount=2
StartOfRange=1000000001
EndOfRange=1000000002
=
#
# Voucher records start
#
1000000001,8986
1000000002,4887
#
# End of voucher records
#
```

CCS3 DES GPG voucher batch example

This text shows the beginning of an example export voucher batch file generated by ccsVoucher_CCS3 using GnuPG encryption (and a bespoke voucher file writer plugin to format the non-header details).

Note: This file has been decrypted using the gpg key.

```
#
# Voucher file for batch 84
# Generated by ccsVoucher at Tue Nov 11 12:58:27 2008
# (key=value or
voucherserialnumber,vouchernumber,vouchersecret,vouchercontext,voucherprivate_secret
)
#
BillingEngineName=PCDEV
VoucherTypeName=DES
AuthRuleName=DES (VL=10 VP=4)
AuthModName=DES
VoucherBatchBatch=
VoucherBatchID=84
```

```
OriginalCount=2
StartOfRange=0000000003
EndOfRange=0000000004
=
#
# Voucher records start
#
1000000136,0000000003,8986
1000000137,0000000004,4887
#
# End of voucher records
#
```

CCS3 CB10 voucher batch example

This text shows an example export voucher batch file generated by ccsVoucher_CCS3 using the 'CB10 HRN' encryption library using the 'HRNGEN' encryption algorithm, but no GnuPG key.

```
#
# Voucher file for batch 85
# Generated by ccsVoucher at Tue Nov 11 12:55:27 2008
# (key=value or voucherbatch,preuseexpiry,hrn,serialnumber)
#
BillingEngineName=PCDEV
VoucherTypeName=CB10
AuthRuleName=CB10 (S=14 R1=2 R2=2 R3=0)
AuthModName=CB10 HRN
VoucherBatchBatch=
VoucherBatchID=85
OriginalCount=2
StartOfRange=00000000000001
EndOfRange=00000000000002
=
#
# Voucher records start
#
85,20090101000000,631599527570333589,1000000138
85,20090101000000,855619036698319621,1000000139
#
# End of voucher records
#
```

CCS3 CB10 GPG voucher batch example

This text shows an example export voucher batch file generated by ccsVoucher_CCS3 using the 'CB10 HRN' encryption library using the 'HRNGEN' encryption algorithm, and GnuPG encryption.

Note: This file has been decrypted using the gpg key.

```
#
# Voucher file for batch 86
# Generated by ccsVoucher at Tue Nov 11 12:55:27 2008
# (key=value or voucherserialnumber,hrnserialnumberseed,hrn,nrnlength,hrnc)
#
BillingEngineName=PCDEV
VoucherTypeName=CB10 HRN
AuthRuleName= CB10 (S=14 R1=2 R2=2 R3=0)
AuthModuleName=CB10 HRN
VoucherBatchBatch=
VoucherBatchID=86
OriginalCount=2
StartOfRange=00000000000003
EndOfRange=00000000000004
=
```

```
#
# Voucher records start
#
86,20090101000000,057195727842702414,1000000138
86,20090101000000,363323157948027866,1000000139
#
# End of voucher records
#
```

Chapter 4

Background Processes

Overview

Introduction

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

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

In this chapter

This chapter contains the following topics.

beVWARS

Purpose

beVWARS is the Vouchers Wallets Accounts Reserve System. It enables CCS to handle actions that interact with the wallet, account, and voucher tables in the E2BE database on the VWS. Most beVWARS functionality is provided by plug-ins and handlers as defined in the *handlers* (on page 34) and plugins parameters. This section shows beVWARS configuration, which includes CCS plug-ins and handlers.

Note: If the VWS is not used, the beVWARS handlers and plug-ins are not relevant.

Location

This binary is located on VWS nodes.

More information

For more information about the beVWARS and its handlers and plugins, see *VWS Technical Guide* and *CCS Technical Guide*.

Example

An example of the beVWARS parameter group of a Voucher and Wallet Server eserv.config file is listed below. Comments have been removed.

```
beVWARS = {
    <other beVWARS configuration>
    handlers = [
        <UBE beVWARS handlers>
        "ccsVWARSVoucherHandler.so"
    ]
}
```

Parameters

beVWARS has one parameter which is relevant to CCS configuration. It is documented below. For more information about other beVWARS parameters, see *VWS Technical Guide* and *CCS Technical Guide*.

`handlers`

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

Description:	Lists the beVWARS message handler plugins to load.
Type:	Array
Optionality:	Required to load handlers which handle messages from CCS processes such as Messaging Manager.
Allowed:	
Default:	
Notes:	This array will also include the standard handlers provided by VWS, and may also include plugins from other applications such as OSA CHAM. For more information about the voucher handler provided with Voucher Manager including its configuration, see *ccsVWARSVoucherHandler* (on page 39).

Example:	`handlers = [`
	`  "ccsVWARSVoucherHandler.so"`
	`]`

ccsBeAvd

Purpose

ccsBeAvd runs on a regular basis as a single independent instance on each VWS node. It deletes a range of archived/deleted voucher records from the VWS.

ccsBeAvd determines which range of archived/deleted vouchers to delete based on the information held in the AVD (archive voucher delete) configuration table. This table holds the range of voucher redeem dates for which vouchers should be deleted and the number of records to delete in one go.

After deleting a range of vouchers, ccsBeAvd deletes the related row from the AVD configuration table.

ccsBeAvd can also delete a single voucher batch based on the information held in the AVD node table.

Replication node registration

ccsBeAvd is registered as a replication node. The replication node id is taken from the -r command line parameter at start up. See *Parameters* (on page 35).

Location

This binary is located on the SMS node.

Startup

ccsBeAvd is started by the cron daemon via a shell script (ccsBeAvdStartup.sh) entry in the crontab.

Example

Here is an example crontab entry for the ccsBeAvd startup script.

```
0 1 * * 0 ( . /IN/service_packages/CCS/ .profile ; . /IN/service_packages/CCS/
.profile ; /IN/service_packages/CCS/bin/ccsBeAvdStartup.sh ) >>
/IN/service_packages/CCS/tmp/ccsBeAvd.log 2>&1
```

Parameters

ccsBeAvd supports the following command line parameters.

`-r`

Syntax:	`-r` *`node_id`*
Description:	*`node_id`* is the reference number which uniquely identifies the ccsBeAvd instance on the VWS as a replication node.
Type:	Integer
Optionality:	Required
Allowed:	A number between 512 and 1023
Default:	None
Notes:	For more information about node numbers, see *SMS Technical Guide*.
Example:	`-r 611`

Failure

If ccsBeAvd fails some of the archived voucher records may not be deleted from the database and the entry for the ccsBeAvd replication node may not be deleted from the AVD configuration table. In this case, the voucher records will be processed the next time ccsBeAvd is run.

Output

ccsBeAvd logs output to the following log file:
`/IN/services_packages/CCS/tmp/ccsBeAvd.log`.

ccsBeAvd

Purpose

ccsSmsAvd runs on a regular basis as a single independent instance on each SMS node. It deletes a range of archived/deleted voucher records from the SMS.

ccsSmsAvd determines which range of archived/deleted vouchers to delete based on the information held in the AVD (archive voucher delete) configuration table. This table holds the range of voucher redeem dates for which vouchers should be deleted and the number of records to delete in one go.

After deleting a range of vouchers, ccsSmsAvd deletes the related row from the AVD configuration table. ccsSmsAvd can also delete a single voucher batch based on the information held in the AVD node table.

The existing configuration file, `/IN/service_packages/CCSVCHRPART/etc/ccs_voucher_reference_part.cfg`, is used to control the archiving functionality of the ccsSmsAvd binary when it deletes a voucher, with the addition of optional flat file archiving of vouchers.

Location

This binary is located in SMS.

Startup

ccsSmsAvd is started by the cron daemon via a shell script (ccsSmsAvdStartup.sh) entry in the crontab.

Example

Here is an example crontab entry for the ccsSmsAvd startup script.

```
0 1 * * 0 ( . /IN/service_packages/CCS/ .profile ; . /IN/service_packages/CCS/
.profile ; /IN/service_packages/CCS/bin/ccsSmsAvdStartup.sh ) >>
/IN/service_packages/CCS/tmp/ccsSmsAvd.log 2>&1
```

Failure

If ccsSmsAvd fails, some of the archived voucher records may not be deleted from the database and the entry for the ccsSmsAvd replication node may not be deleted from the AVD configuration table. In this case, the voucher records will be processed the next time ccsSmsAvd is run.

Output

ccsSmsAvd logs output to the following log file:
`/IN/services_packages/CCS/tmp/`ccsSmsAvd`.log`.

ccsCB10HRNAES

Purpose

The ccsCB10HRNAES security plugin provides the CB10 HRN AES256 encryption functions. *ccsVoucher_CCS3* (on page 38) uses these functions during voucher generation (depending on configuration). It is also used by *ccsVWARSVoucherHandler* (on page 39) during voucher redemption to validate HRN numbers.

For more information about encryption, see *CB-10 HRN Creation* (on page 11).

Location

This binary is located on SMSs, SLCs, and VWSs.

Startup

ccsCB10HRNAES is used by ccsVoucher_CCS3 as necessary. No startup configuration is required to use these libraries.

Configuration

ccsCB10HRNAES has no specific configuration. It does accept some parameters from ccsVoucher_CCS3 for voucher encryption which are configured in the Voucher Management and Service Management screens.

ccsCB10HRNSHA

Purpose

The ccsCB10HRNSHA security plugin provides the CB10 HRN SHA256 and CB10 HRN SHA512 hash functions. *ccsVoucher_CCS3* (on page 38) use these functions during voucher generation (depending on configuration). It is also used by *ccsVWARSVoucherHandler* (on page 39) during voucher redemption to validate HRN numbers.

For more information about encryption, see *CB-10 HRN Creation* (on page 11).

Location

This binary is located on SMSs, SLCs, and VWSs.

Startup

ccsCB10HRNSHA is used by ccsVoucher_CCS3 as necessary. No startup configuration is required to use these libraries.

Configuration

ccsCB10HRNSHA has no specific configuration. It does accept some parameters from ccsVoucher_CCS3 for voucher encryption which are configured in the Voucher Management and Service Management screens.

ccsLegacyPIN

Purpose - ccsLegacyPIN

ccsLegacyPIN plugin library is used by the *ccsVoucher_CCS3* (on page 38) voucher tool to encrypt the voucher's PINs when it generates vouchers using the DES authentication rule. The ccsLegacyPIN plugin library is not applicable to new voucher batches. For more information about authentication rules, see Security libraries.

Note: The ccs3Encryption plugin is a symbolic link to the *ccsLegacyPIN* (on page 38) plugin, but in the ccs3Encryption mode is uses different parameters.

Location

This binary is located on SMSs and VWSs.

Startup - ccsLegacyPIN

ccsLegacyPIN is used by ccsVoucher_CCS3 for existing vouchers. No startup configuration is required for this library to be used.

Configuration

ccsLegacyPIN has no specific configuration. It does accept some parameters from *ccsVoucher_CCS3* (on page 38) for voucher encryption which are configured in the CCS Voucher Management and Service Management screens.

ccsVoucher_CCS1

Purpose

ccsVoucher_CCS1 is used by the ccsChangeVoucherStartup.sh script for creating or changing the state of a range of vouchers. It provides the AltAuthMod PAM for voucher generation.

Location

This binary is located on the SMS node.

Startup

ccsVoucher_CCS1 can be run from the command line. However it is usually started by ccsChangeVoucherStartup.sh in response to an action from the Voucher Management screen.

Configuration

ccsVoucher_CCS1 supports both **eserv.config** parameters and command line parameters. For more information about the configuration available to this process, see *ccsVoucherStartup.sh* (on page 17).

ccsVoucher_CCS3

Purpose

ccsVoucher_CCS3 provides voucher generation functionality for most voucher generation methods.

Location

This binary is located on the SMS node.

Startup

ccsVoucher_CCS3 can be run from the command line. However it is usually started by *ccsVoucherStartup.sh* (on page 17) in response to an action from the Voucher Management screen.

Configuration

ccsVoucher_CCS3 supports both **eserv.config** parameters and command line parameters. For more information about the configuration available to this process, see *ccsVoucherStartup.sh* (on page 17).

Output

For more information about the voucher files written by ccsVoucher_CCS3, see *Exported voucher batch files* (on page 29).

ccsVWARSVoucherHandler

Purpose

This beVWARS message handler performs the Voucher and Wallet Server side processing of messages directly relating to vouchers. This includes voucher reservation/commit, alteration and deletion. It does not perform the wallet recharge; this is done by the ccsVWARSWalletHandler. The message handler only controls the Voucher and Wallet Server side of the CCS voucher tables, not the main body of data about vouchers that is replicated from the SMS.

This handler validates incoming voucher reserve (for example, scratch or redeem) requests, and refers to the replicated CCS voucher tables for all information except the current redeemed/unredeemed state of the voucher.

It is important to remember that the BE_VOUCHER record will in all probability not exist unless the voucher has had a previous successful (or almost successful) redeem performed upon it. This state is hidden from the client process, a non-existent BE_VOUCHER record is proof that the voucher has not been redeemed.

Location

This binary is located on VWS nodes.

Startup

If ccsVWARSVoucherHandler is included in the beVWARS `handlers` array in **eserv.config**, it is loaded by beVWARS when beVWARS is initialized.

It is included in the following lines:

```
handlers = [
    "ccsVWARSVoucherHandler.so"
]
```

For more information about the beVWARS `handlers` section, see *handlers* (on page 34).

Note: Other handlers may also be included in the `handlers` array.

Parameters

The ccsVWARSVoucherHandler supports the following parameters in the `beVWARS` section of **eserv.config** .

Note: It also required the `BE.serverId` parameter. For more information about setting `serverId`, see *VWS Technical Guide*.

`badPinExpiryHours`

Syntax:	`badPinExpiryHours = hours`
Description:	The number of hours before the bucket storing the bad PIN expires.
Type:	Integer
Optionality:	Optional (default used if not set)
Allowed:	negative integer Does not expire positive integer Number of hours before expiry
Default:	24
Notes:	
Example:	`badPinExpiryHours = 48`

`consecutiveBadPinExpiryHours`

Syntax:	`consecutiveBadPinExpiryHours = hours`
Description:	The number of hours before the bucket storing the consecutive bad PIN expires.
Type:	Integer
Optionality:	Optional (default used if not set)
Allowed:	negative integer Does not expire positive integer Number of hours before expiry
Default:	24
Notes:	
Example:	`consecutiveBadPinExpiryHours = 48`

`createRechargeCDRInactiveAccount`

Syntax:	`createRechargeCDRInactiveAccount = true|false`
Description:	When true, failed voucher recharges generate an EDR.
Type:	Boolean
Optionality:	Optional (default used if not set)
Allowed:	true, false
Default:	true
Notes:	
Example:	`createRechargeCDRInactiveAccount = true`

`dailyBadPinExpiryHours`

Syntax:	`dailyBadPinExpiryHours = hours`
Description:	The number of hours before the bucket storing the daily bad PIN expires.
Type:	Integer
Optionality:	Optional (default used if not set)
Allowed:	negative integer Does not expire positive integer Number of hours before expiry

Default: 24
Notes:
Example: `dailyBadPinExpiryHours = 48`

monthlyBadPinExpiryHours

Syntax: `monthlyBadPinExpiryHours = hours`
Description: The number of hours before the bucket storing the monthly bad PIN expires.
Type: Integer
Optionality: Optional (default used if not set)
Allowed:

negative integer	Does not expire
positive integer	Number of hours before expiry

Default: 744
Notes:
Example: `monthlyBadPinExpiryHours = 744`

requireBonusRow

Syntax: `requireBonusRow = true|false`
Description: When true, vouchers will fail if there is no entry in CCS_BONUS_VALUES.
Type: Boolean
Optionality: Optional (default used if not set)
Allowed: true, false
Default: true
Notes:
Example: `requireBonusRow = true`

updateLastUseVoucherRecharge

Syntax: `updateLastUseVoucherRecharge = true|false`
Description: When true, voucher recharges update the 'last use date' field.
Type: Boolean
Optionality: Optional (default used if not set)
Allowed: true, false
Default: true
Notes:
Example: `updateLastUseVoucherRecharge = true`

vomsInstalled

Syntax: `vomsInstalled = true|false`
Description: Define if you are using:
- Voucher Manager-type bad PIN balances (true)
- Just a single, VWS bad PIN (false)

Type: Boolean
Optionality: Optional (default used if not set)
Allowed: true, false
Default: false

Notes:

Example: `vomsInstalled = true`

`replicationInterface`

Syntax: `replicationInterface = "if"`

Description: The handle of the SLEE replication interface.

Type: String

Optionality: Optional (default used if not set)

Allowed: Must match the Interface name in **SLEE.cfg**.

Default: "replicationIF"

Notes: For more information about **SLEE.cfg**, see *SLEE Technical Guide*.

Example: `replicationInterface = "replicationIF"`

Example

An example of the `voucherHandler` parameter group of a Voucher and Wallet Server **eserv.config** file is listed below. Comments have been removed.

```
voucherHandler = {
    requireBonusRow = true
    updateLastUseVoucherRecharge = true
    createRechargeCDRInactiveAccount = true
    badPinExpiryHours = 24
    dailyBadPinExpiryHours = 24
    monthlyBadPinExpiryHours = 744
    consecutiveBadPinExpiryHours = -1

    vomsInstalled = true
    replicationInterface = "replicationIF"
}
```

Failure

If ccsVWARSVoucherHandler fails, interaction with the wallets from the SLC involving vouchers will fail.

Output

The ccsVWARSVoucherHandler writes error messages to the system messages file, and also writes additional output to **/IN/service_packages/E2BE/tmp/beVWARS.log**.

ccsVoucherCreationPlugin

Purpose

The ccsVoucherCreationPlugin library is used by *ccsVoucher_CCS3* (on page 38) to generate the headers and footers of voucher batch files.

Location

This binary is located on the SMS node.

Startup

ccsVoucherCreationPlugin is used by ccsVoucher_CCS3 as necessary. No startup configuration is required for this library to be used.

Configuration

ccsVoucherCreationPlugin has no specific configuration. It does accept some parameters from ccsVoucher_CCS3 for voucher encryption which are configured in the Voucher Management and Service Management screens.