Oracle® Communications Convergent Charging Controller

Provisioning Interface User's and Technical Guide

Release 15.0.0

October 2023

Copyright

Contents

About This Document

Scope

The scope of this document includes all the information required to install, configure and administer the provisioning interface (PI) application. It does not include detailed design of the service.

Audience

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

Prerequisites

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

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

Related Documents

The following documents are related to this document:

- *ACS Provisioning Interface Commands*
- *CCS Provisioning Interface Commands*
- *MM Provisioning Interface Commands*
- *NP Provisioning Interface Commands*
- *VPN Provisioning Interface Commands*
- *Charging Control Services Technical Guide*
- *Service Management System Technical 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.

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 to the Provisioning Interface

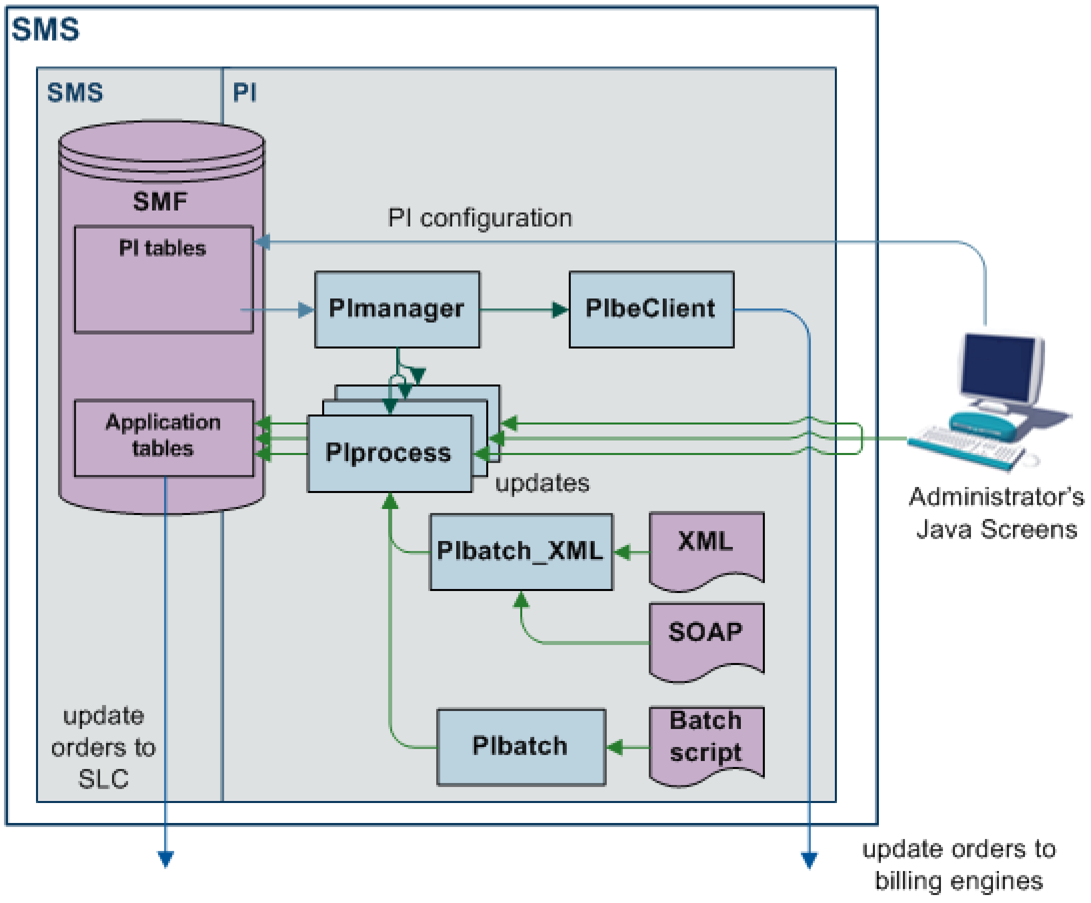
Introduction

The Provisioning Interface (PI) provides a mechanism for manipulating data in the SMF database using an API. It enables bulk or scripted operations on the SMF data to be completed, where a human operator using the Java administration screens would be inefficient or error-prone.

The PI provides a reliable, extensible, network aware interface based on interoperability standards (for example XML).

Component diagram

This diagram shows the PI components and processes.



Component descriptions

This table describes the main components involved in the Provisioning Interface application.

Component	Description	Further Information
SMF database	The main SMF database holds the configuration details which are updated by the PI and some PI configuration details. The SMF holds configuration data for: • The system • Client accounts • Services	*SMS Technical Guide*
PI Administration screens	Enables an administrator to interact with the SMF database. The PI screens enable you to add new PI commands, users and hosts to the system and send test messages to specificPI processes.	*PI Administration Screen* (on page 47)
PI Tester screens	Test the system by sending individual commands to PI processes.	*PI Tester Screen* (on page 67)
PImanager	Starts and stops PI processes. PImanager will start as many PIprocesses as specified in the PI_PORTS table on the SMF database.	*PImanager* (on page 75)

Component	Description	Further Information
PIprocess	Runs on an SMS listening at a specific port for PI commands.	*PIprocess* (on page 77)
PIbatch	Sends multiple PI commands to the PIprocesses. PIbatch can take instructions from a batch file, enabling complex treatments of the data in the SMF database to take place.	*PIbatch* (on page 78)
PIbatch_XML	Sends multiple PI commands to the PIprocesses. PIbatch_XML takes instructions from XML and SOAP files.	*PIbatch XML* (on page 80)
eserv.config	The configuration file for PI. **Note:** Not all installations require this file to be configured.	**eserv.config Configuration** (on page 6)

Process

This table describes the process involved in running a PI command.

Note: The security/authentication parts of this process will only happen if the security plug-in is active in your deployment.

Step	Action
1	The first message sent to the server to start a new connection is a login message. **Note:** All communication between the client and the SMS uses the ASCII, HTTP/1.1, or HTTPS/1.1 protocol over TCP/IP. This enables all messages to be passed on a single connection for the duration of the session.
2	On successful login, the server will optionally send a security token. If security is used, then this token should be included in future request messages to confirm authentication.
3	A client system sends PI command to the relevant network port on the SMS.
4	The server process will check the authentication state, if configured to do so, and pass the requested command to the appropriate handler.
5	The server will respond on the same network connection with a message containing the response. The response will consist of any returned data, and, if configured, the new security token for use in future messages to the PI.
6	An end session command will be sent from either the server or the client to the other party.
7	The session is closed.

Triggering BPL tasks

This process describes how a PI command triggers a BPL task.

For more information about BPL tasks, see *CCS User's Guide* and *CCS Technical Guide*.

Stage	Description
1	A client system sends the CCSBPL command to the relevant network port on the SMS.
2	The *PIprocess* (on page 77) for that port calls the libPI_CCSBPL library and sends the request to the smsTrigDeamon to trigger the BPL task process.

Stage	Description
	For more information about the BPL task process, see *SMS Technical Guide*.
3	When the BPL has been processed, the smsTrigDaemon returns the result of the command to PIprocess.
4	PIprocess translates the response into a PI command response for the CCSBPL command and returns it to the client system.

PI Commands

Introduction

The provisioning interface uses TCP/IP-based UNIX sockets to receive provisioning commands and parameters.   These are translated into SQL commands that update prepaid application tables of the SMF and E2BE Oracle databases.

Note: The output from the PI command is limited to 2,000 characters. When the output exceeds this limit, the output is truncated to the "<command>:ACK" message.

PI command installation

The PI commands which are available depend on which packages were run when the PI was installed. For details about the commands available for your installation, see the PI commands guide for your applications.

Example: For the commands for CCS, see *CCS Provisioning Interface Commands*.

Command package details

This table shows the functionality installed by each package.

Functionality	Required	Package
Framework to execute a PI command.	Required	piSms
Core CCS commands and VWS client.	Required	piCcsSms
Command definitions for a subscriber domain.	Optional	piSubscriberSms
Command definitions for a wallet domain.	Optional	piWalletSms
Command definitions for a voucher domain.	Optional	piVoucherSms
Command definitions for the Social Networking Service Template.	Optional	piSrmSms

For more information about installing these packages, see *Installation and Removal Overview* (on page 91).

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.

Configuration Overview

Introduction

Most of the configuration required to set up the PI is completed automatically when the packages are installed, or when the configuration scripts are run. However, some tasks must be completed by hand after the packages have been installed.

Configuration components

This table describes the configuration required to configure the PI.

Component	Description	Further information
eserv.config	This file provides a centralized location for configuring Convergent Charging Controller software, including PI. This file should be updated with any relevant details from the **eserv.config.pi_example** file added during installation.	*eserv.config Configuration* (on page 6)
PI administration screens	PI uses Java screens to administer user accounts, connections and commands. These screens will be populated with data entered during the installation, but may require additional configuration.	*PI Administration Screen* (on page 47)

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.

eserv.config.pi_example

The PI comes with an example of the PI's **eserv.config** configuration in a file called **eserv.config.pi_example** in the root of the application directory.   This example configuration should be copied into the main **eserv.config** file to provide a base for the application's configuration.

Optional sections in eserv.config

Some sections of the **eserv.config** file are only required if your deployment has a specific component. For example, the `PIbeClient` section is required only if a VWS Voucher and Wallet Server is used.

eserv.config subsections

The `pi` section of the **eserv.config** file has the following structure.

```
pi = {
    general = {}

    authentication = {}

    throttling = {}

    PIbeClient = {}

    ssl = {}

    soap = {}
}
```

Parameters for each subsection are described below.

General

Here is an example of the general sub section of the PI **eserv.config** configuration.

```
    general = {
        # debug = 'N'
        # oraUser = "/"
        # synstamp = 'Y'
        # timeout = 30
        # logLevel = 0
        # securityPlugin = ""
        # correlationRequestTagName = "CORRELATE"
        # correlationResponseTagName = "CORRELATE"
    }
```

The parameters are described in detail below.

`coreWhenProcessUnresponsive`

Syntax:	`coreWhenProcessUnresponsive = true|false`
Description:	Specifies how the PImanager process responds when PIprocess is unresponsive.
Type:	Boolean
Optionality:	Optional (default used if not set)
Allowed:	• true – PImanager sends a SIGABRT signal when PIprocess is unresponsive. • false – PImanager sends a SIGTERM signal when PIprocess is unresponsive.
Default:	False
Notes:	
Example:	`coreWhenProcessUnresponsive = true`

`correlationRequestTagName`

Syntax:	`correlationRequestTagName = "`*`name`*`"`
Description:	The expected correlation tag in the XML message.
Type:	String
Optionality:	Optional (default used if not set).
Allowed:	
Default:	`"CORRELATE"`
Notes:	
Example:	`correlationRequestTagName = "CORRELATE"`

`correlationResponseTagName`

Syntax:	`correlationResponseTagName = "`*`name`*`"`
Description:	The expected correlation tag in the XML response.
Type:	String
Optionality:	Optional (default used if not set).
Allowed:	
Default:	'correlationRequestTagName' value
Notes:	If not set it will be the value on 'correlationRequestTagName'.
Example:	`correlationResponseTagName = "CORRELATE"`

`debug`

Syntax:	`debug = `*`Y|N`*
Description:	Turn debug on or off.
Type:	String
Optionality:	Optional (default used if not set).
Allowed:	
Default:	N
Notes:	Only turn on in extreme circumstances as it turns on debug for all parts of PI. Command line equivalent: "`-D`" (Y), missing (N).
Example:	`debug = 'N'`

`loglevel`

Switches the PIprocess trace on or off.

Default:	0	
Allowed:	1	switches the PIprocess trace on.
	0	switches the PIprocess trace off

`oraUser`

Defines the Oracle user name and password.

Default:	"/"
Note:	This parameter is not usually required as the default "/" is correct.
	Command line equivalent: "`-u /`"

`securityPlugin`

The name of any security plug-in used.

Default:	""
Allowed:	string

`synstamp`

Turns the synstamp on or off.

Default:	Y	
Allowed:	Y	turns synstamp on
	N	turns synstamp off
Note:	Command line equivalent of "`-S Y`".	

`timeout`

The maximum allowed time, in seconds, for a PI command. If, for example, you set `timeout = 30`, timeout will occur after 30 seconds.

Default:	-1	
Allowed:	-1	No timeout.
	positive integer	Seconds before timeout.
Note:	The command line equivalent would be "`-t 30`".	

authentication

The authentication subsection of the PI eserv.config configuration supports these parameters.

```
authentication = {
    timeout = 0
    noAuthTokenForAnyPIError = false
}
```

The parameter is described in detail below.

`timeout`

Syntax:	`timeout =` *`seconds`*
Description:	Authentication token timeout.
Type:	Integer
Optionality:	Optional (default used if not set).
Allowed:	

Default:	0 (no timeout)
Notes:	
Example:	`timeout = 0`

`noAuthTokenForAnyPIError`

Syntax:	`noAuthTokenForAnyPIError = ` *`True`* ` | ` *`False`*	
Description:	Specifies whether the PI returns an authentication token when an error condition occurs.	
Type:	Boolean	
Optionality:	Optional	
Allowed:	true	The PI does not return an authentication token when an error condition occurs.
	false	The PI returns an authentication token when an error condition
Default:	false	
Notes:		
Example:	`noAuthTokeknForAnyPIError = False`	

Throttling

The throttling subsection of the PI **eserv.config** configuration supports these parameters.

```
throttling = {
    sendRate = 0<int>
}
```

The parameter is described in detail below.

`sendRate`

The maximum number of PI commands per second.

Default:	0	
Allowed:	0	no limit
	positive integer	commands per second.

PIbeClient

The `PIbeClient` section of the **eserv.config** file configures the *PIbeClient* (on page 78) process. Here is the structure of the section.

```
PIbeClient = {
    namedEventCanSendDebitBalanceNegative = 'n|y'
    oracleLogin = "usr/pwd"

    beLocationPlugin = "lib"
    clientName = "name"

    heartbeatPeriod = microsecs
    messageTimeoutSeconds = seconds
    maxOutstandingMessages = int
    reportPeriodSeconds = seconds
    connectionRetryTime = seconds

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

        confStr = {
            plug-in_configuration
        }

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

The parameters are described in detail below.

clientName

Syntax:	`clientName = "name"`
Description:	The unique client name of the process.
Type:	String
Optionality:	Required
Allowed:	Must be unique
Default:	The hostname of the local machine.
Notes:	The server generates clientId from a hash of *name*. 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 = "PIbeClient"`

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
Allowed:	
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:	`maxOutstandingMessages =` *`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:	`maxOutstandingMessages = 100`

messageTimeoutSeconds

Syntax:	`messageTimeoutSeconds =` *`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:	`messageTimeoutSeconds = 2`	

namedEventCanSendDebitBalanceNegative

Determines whether a named billable event charge can be sent a negative debit balance.

Default:	`n`	
Allowed:	y	negatives will be allowed
	Y	negatives will be allowed
	n	only positives will be allowed

notEndActions

Syntax:	`notEndActions = [`
	`    {type="str", action="[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:	see below

```
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"}
]
```

action

Syntax:	
Description:	Action to take with a message.
Type:	
Optionality:	
Allowed:	• "NACK" • "ACK"
Default:	
Notes:	
Example:	

type

The type of message.

oracleLogin

Identifies the PIbeClient when it logs on to the database.

Default: "/"

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:	
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. This parameter is used by libBeClientIF.
Example:	`plugins = [ { config="broadcastOptions" library="libclientBcast.so" function="makeBroadcastPlugin" } ]`

Syntax:

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

Example:

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

config

Syntax:	`config="`*`name`*`"`
Description:	The name of the configuration section for this plug-in. This corresponds to a configuration section within the `plugins` section in the **eserv.config** file.
Type:	String
Optionality:	Required (must be present to load the plug-in)
Allowed:	
Default:	No default
Notes:	
Example:	`config="voucherRechargeOptions"`

function

Syntax:	`function="`*`str`*`"`
Description:	The function the plug-in should perform.
Type:	String
Optionality:	Required (must be present to load the plug-in)
Allowed:	
Default:	No default
Notes:	
Example:	`function="makeVoucherRechargePlugin"`

`library`

Syntax:	`library="`*`lib`*`"`
Description:	The filename of the plug-in library.
Type:	String
Optionality:	Required (must be present to load the plug-in)
Allowed:	
Default:	No default
Notes:	
Example:	`library="libccsClientPlugins.so"`

Voucher and wallet plugins

There are four plug-ins which provide functionality for the PIbeClient:

1 Voucher recharge (VRW)
2 Voucher type recharge (VTR)
3 Merge wallets (MGW)
4 *Broadcast* (on page 20)

Note: The broadcast plug-in configuration must be placed last in the `plugins` configuration section.

Each plug-in can have a configuration section. The name of this subsection will match the string provided for the config parameter in the `plugins` subsection.

Example: The Voucher Recharge plug-in has config set to `voucherRechargeOptions`. So the configuration section for this plug-in is:

```
voucherRechargeOptions = {
    ...
}
```

`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`. This parameter is used by libBeClientIF.
Example:	`reportPeriodSeconds = 10`

Voucher Recharge plug-in

The Voucher Recharge BeClient plug-in executes voucher recharges.

The `plugins` section must include the following configuration to load this plug-in.

```
{
    config="voucherRechargeOptions",
    library="libccsClientPlugins.so",
    function="makeVoucherRechargePlugin"
}
```

Note: The VRW plug-in requires the broadcast plug-in.

The voucher recharge plug-in supports the following configuration.

```
voucherRechargeOptions = {
    srasActivatesPreuseAccount=true|false
    voucherServerCacheLifetime = seconds
    voucherServerCacheCleanupInterval = seconds
    sendBadPin = true|false
    voucherRechargeTriggers = [
        "str"
    ]
}
```

The parameters are described in detail below.

`sendBadPin`

Syntax:	`sendBadPin = true|false`
Description:	Whether or not to increment the Bad PIN count for a failed voucher redeem.
Type:	Boolean
Optionality:	Optional
Allowed:	• true – Increment Bad PIN count for each failed attempt to recharge a voucher. • false – Do not increment Bad PIN count for failed attempts to recharge a voucher.
Default:	false
Notes:	This parameter: • applies only to an invalid voucher number or voucher PIN. It does not apply to failed wallet recharges • is part of the `voucherRechargeOptions` parameter group
Example:	`sendBadPin = false`

`srasActivatesPreuseAccount`

Syntax:	`srasActivatesPreuseAccount = true|false`
Description:	Sets whether or not alternate subscribers can activate subscriber accounts which are in a pre-use state.
Type:	Boolean
Optionality:	Optional
Allowed:	• true – A scratch card alternate subscriber can activate a pre-use account. • false – A scratch card alternate subscriber cannot activate a pre-use account.
Default:	true
Notes:	This parameter is: • Not used by ccsBeOrb • Part of the `voucherRechargeOptions` parameter group

Example:	`srasActivatesPreuseAccount = false`

voucherRechargeTriggers

Syntax:

```
voucherRechargeTriggers = [
    "VRW "
]
```

Description:	This message triggers the voucher recharge plug-in.
Type:	Array
Optionality:	Required
Allowed:	VRW
Default:	
Notes:	This parameter array is part of the `voucherRechargeOptions` parameter group.
Example:	

voucherServerCacheCleanupInterval

Syntax:	`voucherServerCacheCleanupInterval =` *`seconds`*
Description:	Time in seconds between purges of the voucher server id cache.
Type:	Integer
Optionality:	Optional
Allowed:	Any positive decimal integer.
Default:	60 (seconds)
Notes:	
Example:	`voucherServerCacheCleanupInterval = 60`

voucherServerCacheLifetime

Syntax:	`voucherServerCacheLifetime =` *`seconds`*
Description:	Time in seconds to hold items in the voucher server ID cache.
Type:	Integer
Optionality:	Optional
Allowed:	Any positive decimal integer.
Default:	600 (seconds)
Notes:	
Example:	`voucherServerCacheLifetime = 600`

Voucher Type Recharge plug-in

The Voucher Type Recharge PIbeClient plug-in executes voucher type recharges.

The `plugins` section must include the following configuration to load this plug-in.

```
{
    config="voucherTypeRechargeOptions",
    library="libccsClientPlugins.so",
    function="makeVoucherTypeRechargePlugin"
}
```

Note: The VTR plug-in requires the broadcast plug-in.

The voucher recharge plug-in supports the following configuration.

```
        voucherTypeRechargeOptions = {
            srasActivatesPreuseAccount=true|false
            voucherTypeRechargeTriggers = [
                "VTR "
            ]
        }
```

The parameters are described in detail below.

`srasActivatesPreuseAccount`

Syntax:	`srasActivatesPreuseAccount = true|false`
Description:	Sets whether or not alternate subscribers can activate subscriber accounts which are in a pre-use state.
Type:	Boolean
Optionality:	Optional
Allowed:	• true – A scratch card alternate subscriber can activate a pre-use account. • false – A scratch card alternate subscriber cannot activate a pre-use account.
Default:	true
Notes:	This parameter is: • Not used by ccsBeOrb • Part of the `voucherRechargeOptions` parameter group
Example:	`srasActivatesPreuseAccount = false`

`voucherTypeRechargeTriggers`

Syntax:	`voucherTypeRechargeTriggers = [` `    str [...]"` `]`
Description:	Starts the voucher type recharge plug-in.
Type:	Array
Optionality:	Required
Allowed:	VRW
Default:	
Notes:	This parameter array is part of the `voucherTypeRechargeOptions` parameter group.
Example:	`voucherTypeRechargeTriggers = ["VTR "]`

Merge Wallets plug-in

The Merge Wallets PIbeClient plug-in executes wallet merges.

The `plugins` section must include the following configuration to load this plug-in.

```
            {
                config = "mergeWalletsOptions",
                library = "libccsClientPlugins.so",
                function = "makeMergeWalletsPlugin"
            }
```

Note: The VTR plug-in requires the broadcast plug-in.

The merge wallets plug-in supports the following configuration.

```
        mergeWalletsOptions = {
            oracleLogin = "usr/pwd"
            mergeBucketExpiryPolicy = "str"
            mergeWalletExpiryPolicy = "str"
            allowedSourceWalletStates = "str[,...]"
            mergeWalletsTriggers = [
                "str [...]"
            ]
        }
```

The parameters are described in detail below.

allowedSourceWalletStates

Syntax:	`allowedSourceWalletStates = "str[...]"`
Description:	The states the source wallet must be in to allow it to be merged with another wallet.
Type:	String
Optionality:	Required
Allowed:	P Pre-use
	A Active
	D Dormant
	S Suspended
	F Frozen
	T Terminated
Default:	None
Notes:	At least one state must be included, or all merged will be disallowed.
Example:	`allowedSourceWalletStates = "PA"`

mergeBucketExpiryPolicy

Syntax:	`mergeBucketExpiryPolicy = "str"`
Description:	Determines how the bucket expiry policy is treated.
Type:	String
Optionality:	Optional (default used if not set).
Allowed:	merge policy is merged
	move policy is moved
Default:	merge
Notes:	
Example:	`mergeBucketExpiryPolicy = "move"`

mergeWalletExpiryPolicy

Syntax:	`mergeWalletExpiryPolicy = "str"`
Description:	Determines the way expiry dates for merged wallets are managed.
Type:	String
Optionality:	Optional
Allowed:	best The expiry date of the wallet with the most time left is used.
	ignore The expiry date of the source wallet is ignored.
Default:	best
Notes:	
Example:	`mergeWalletExpiryPolicy = "best"`

mergeWalletsTriggers

Syntax:	`mergeWalletsTriggers = [`
	` "str [...]"`
	`]`

Description:	Wallets of this type starts the merge wallets plug-in.
Type:	Array of strings.
Optionality:	Required
Allowed:	MGW
Default:	None
Notes:	The syntax must be typed exactly as shown in the example.
Example:	`mergeWalletsTriggers = ["MGW "]`

oracleLogin

Syntax:	`oracleLogin = "usr/pwd"`
Description:	The login details the BeClient should use to log in to the SMF database, when performing merge wallet functions.
Type:	String
Optionality:	Optional
Allowed:	
Default:	/
Notes:	
Example:	`oracleLogin = "smf/smf"`

Broadcast plug-in

The Broadcast PIbeClient plug-in overrides the beLocationPlugin that would normally load connection details from the database.

The `plugins` section must include the following configuration to load this plug-in.

```
{
    config="",
    library="libccsClientPlugins.so",
    function="makeBroadcastPlugin"
}
```

Notes:

- This plug-in must be the last in the `plugins` subsection.
- This plug-in has no configuration.
- The broadcast plug-in is required by the VRW and VTR plug-ins.

notEndActions

The state conversions subsection supports the following parameter.

```
notEndActions = [
    {type="str", action="str"}
    ...
]
```

The parameter is described in detail below.

`notEndActions`

Syntax:	`notEndActions = [`
	 `    {type="str", action="[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"}`
	 `]`

voucherStateConversions

The state conversions subsection supports the following parameter.

```
voucherStateConversions = {
    str = "ESCHER"[,
    ...]
}
```

The parameter is described in detail below.

`voucherStateConversions`

Syntax:	`voucherStateConversions = {`
	 `    str = "ESCHER"[,`
	 `    ...]`
	 ` }`

Description:	Converts from ESCHER encoding to a single character and back.
Type:	Array
Optionality:	
Allowed:	
Default:	
Notes:	

Example:	`voucherStateConversions = {`
	`    A = "ACTV",`
	`    F = "FRZN",`
	`    R = "RDMD"`
	`}`

stateConversions

The state conversions subsection supports the following parameter.

```
stateConversions = {
    str = "ESCHER"[,
    ...]
}
```

The parameter is described in detail below.

stateConversions

Syntax:	`stateConversions = {`
	`    str = "ESCHER"[,`
	`    ...]`
	`}`

Description:	Converts from ESCHER encoding to a single character and back.
Type:	Array
Optionality:	
Allowed:	
Default:	
Notes:	

Example:	`stateConversions = {`
	`    A = "ACTV",`
	`    P = "PREU",`
	`    D = "DORM",`
	`    F = "FROZ",`
	`    S = "SUSP",`
	`    T = "TERM"`
	`}`

billingEngines

The `billingEngines` subsection supports the following configuration.

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

This section overrides connection details that beLocationPlugin obtains from the database. It identifies the Voucher and Wallet Servers and assigns their Internet connection details.

Note: This section is optional, and is often commented out.

The parameters are described in detail below.

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:	
Default:	
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 (IP) address and associated port number of the primary Voucher and Wallet Server.
Type:	Parameter array
Optionality:	Required if this section is used
Allowed:	
Default:	
Notes:	This parameter is part of the `billingEngines` parameter array.

Examples:	`primary = { ip="192.0.2.0", port=1500 }`
	`primary = { ip = "2001:db8:0000:1050:0005:0600:300c:326b", port=1500 }`
	`primary = {ip = "2001:db8:0:0:0:500:300a:326f", port=1500 }`
	`primary = { ip = "2001:db8::c3", port=1500 }`

`secondary`

Syntax:	`secondary = { ip="`*`ip`*`", port=`*`port`*` }`
Description:	The `secondary` parameter group defines the Internet Protocol (IP) address and associated port number of the secondary Voucher and Wallet Server.
Type:	Array
Optionality:	Required, if this section is used
Allowed:	
Default:	
Notes:	This parameter is part of the `billingEngines` parameter array.

Examples:	`secondary = { ip="192.0.2.1", port=1500 }`
	`secondary = { ip = "2001:db8:0000:1050:0005:0600:300c:326b", port=1500 ]`
	`secondary = {ip = "2001:db8:0:0:0:500:300a:326f", port=1500 }`
	`secondary = { ip = "2001:db8::c3", port=1500 }`

`ip`

Syntax:	`ip = "`*`ip`*`"`
Description:	The Internet Protocol (IP) address of the Voucher and Wallet Server.
Type:	String
Optionality:	Required
Allowed:	IP version 4 (IPv4) addresses, IP version 6 (IPv6) addresses
Default:	None

Notes:	This parameter is part of either the primary, or the secondary parameter group of the `billingEngines` parameter array. You can use the industry standard for omitting zeros when specifying IPv6 addresses.

Examples:	`ip = "192.0.2.0"`
	`ip = "2001:db8:0000:1050:0005:0600:300c:326b"`
	`ip = "2001:db8:0:0:0:500:300a:326f"`
	`ip = "2001:db8::c3"`

`port`

Syntax:	`port = ` *`port`*
Description:	The port number associated with the address of the Voucher and Wallet Server.
Type:	Integer
Optionality:	Required
Allowed:	
Default:	None
Notes:	This parameter is part of either the primary or secondary parameter group of the `billingEngines` parameter array.
Example:	`port = 1500`

ssl

Here is an example of the `ssl` subsection of the PI **eserv.config** configuration.

```
    ssl = {

        allowINSECURESSLv3 = false
        certificateFile = "/IN/service_packages/PI/my_sslCertificate.pem"
        keyFile = "/IN/service_packages/PI/my_sslKey.pem"
    }
```

The parameters in this subsection are described in detail below.

`allowINSECURESSLv3`

Syntax:	`allowINSECURESSLv3 = true|false`
Description:	Whether to allow use of SSLv3 in the SSL handshake for SSL enabled systems. For example, set this parameter to true for customers with an ASP that must use the SSLv3 protocol version. Use of SSLv3 and SSLv2 is disabled by default.
Type:	Boolean
Optionality:	Optional (default used if not set)
Allowed:	• true – Use of SSLv3 protocol version enabled. • false – Use of SSLv3 protocol version disabled.
Default:	false
Notes:	The `allowINSECURESSLv3` parameter can be set for the DAP, PI and OSD components. You should set `allowINSECURESSLv3` to true if the ASP is able to use only SSLv3 protocol version. Otherwise set `allowINSECURESSLv3` to false.
Example:	`allowINSECURESSLv3 = true`

certificateFile

Syntax:	`certificateFile = "`*`filename`*`"`
Description:	The file name of the PEM Base64 encoded DER certificate to be used when accepting HTTPS connections.
Type:	String
Optionality:	Optional – only required if the interface used is SOAP/HTTPS.
Allowed:	
Default:	None
Notes:	
Example:	`certificateFile = "/IN/service_packages/PI/sslCertificate.pem"`

keyFile

Syntax:	`keyFile = "`*`filename`*`"`
Description:	The file name of the private key used to create the certificate.
Type:	String
Optionality:	Optional – only required if the interface used is SOAP/HTTPS.
Allowed:	
Default:	None
Notes:	
Example:	`keyFile = "/IN/service_packages/PI/privKey.pem"`

soap

Note: The template WSDL files for the CCS, ACS, VPN, and NP PI commands are installed in the **/IN/service_packages/PI/etc** directory on the SMS server. You use the WSDL files when developing or configuring clients to allow them to generate the SOAP PI commands that are defined by the WSDL files. Update the template WSDL files with the PI server information (IP address and ports) before you distribute or publish them. After updating the template WSDL files, you may publish the WSDL files on the SMS server using a HTTP Server in an accessible directory; for example, **/IN/html**.

Here is an example structure of the `soap` subsection of the PI **eserv.config** configuration.

```
soap = {

    implicitLoginsSupported = false

    validateAuthStrings = true

    expansionRules = [

        {
            command = "CCSCD1"
            action = "QRY"
            parameter = "BALANCES"
            itemName = "BALANCE_ITEM"
            itemSeparator = "|"
            elementSeparator = ":"
            elementNames = [ "BALANCE_TYPE_NAME", "*BUCKETS" ]
        }
        {
            next_rule_parameters
        }
```

```
            {
                next_rule_parameters
            }
            {
                next_rule_parameters
            }
        ]

    }
```

These parameters are described in detail below.

`validateAuthStrings`

Syntax:	`validateAuthStrings = true|false`
Description:	Turning this variable false will bypass checking of AUTH strings.
Type:	Boolean
Optionality:	Optional (default used if not set).
Allowed:	
Default:	true
Notes:	That is useful in situations where a pool of connections is used for access to PI web services and the clients wish to be able to use any open connection for whichever command they wish execute next, regardless of which one they used for the preceding request.
Example:	`validateAuthStrings = true`

`implicitLoginsSupported`

Syntax:	`implicitLoginsSupported = true|false`
Description:	Turning the following variable true will allow any incoming SOAP request to contain username and password. If such a request is received on an unauthenticated connection, the fields will be used to do an implicit Login (just like a real Login only no LoginResponse is sent). If the login fails a fault is returned; if it succeeds the command is executed.
Type:	Boolean
Optionality:	Optional (default used if not set).
Allowed:	
Default:	false
Notes:	That is useful in situations where a pool of connections is used for access to PI web services and the clients wish to be free from knowing whether or not a particular connection requires authentication before use.
Example:	`implicitLoginsSupported = false`

`expansionRules`

By default when returning PI responses in SOAP format, PI assembles them simplistically by using the parameter name as the tag name, and the value as the tag value. For example the name/value pair MSISDN="1234" is rendered

```
    <pi:MSISDN>1234</pi:MSISDN>
```

However, some commands return complex, repeating, nested elements in a single response parameter, and the customer may wish to configure so-called expansion rules that cause these to be exploded out into a more XML-like and accessible style.

For any parameter of any command, an expansion rule can be configured, to explain how PI should unpack its value (normally by tokenizing on some separator such as "|"). The expansion rule concept allows for the situation where a list of struct-like items are assembled, with list items being separated by one sort of delimiter (e.g. "|") and the fields within each list item being separated by another (e.g. ":").

A rule is also allowed to treat an element as something which itself requires expansion, as shown in the `CCSCD1=QRY BALANCES` rule below.

Note: All these examples are real world in the sense that they can help provide a more usable rendering of the parameter values returned by PI in each case, but it is a matter of customer preference as to whether or not they are activated at a site.

If you choose to use or update the `expansionRules` configuration and if you are using SOAP integration, update your published or distributed WSDL files to match the modified output format of the response. For more information see *soap* (on page 25).

PI command parameters, present or future, may require similar rules (or different ones).

Here is an example of the expansion rules sub-section.

```
expansionRules = [

    {
        command = "CCSCD1"
        action = "QRY"
        parameter = "BALANCES"
        itemName = "BALANCE_ITEM"
        itemSeparator = "|"
        elementSeparator = ":"
        elementNames = [ "BALANCE_TYPE_NAME", "*BUCKETS" ]
    }
    {
        command = "CCSCD1"
        action = "QRY"
        parameter = "*BUCKETS"
        itemName = "BUCKET_ITEM"
        itemSeparator = "|"
        elementSeparator = ":"
        elementNames = [
            "BUCKET_VALUE",
            "BUCKET_EXPIRY"
        ]
    }
    {
        command = "CCSCD7"
        action = "QRY"
        parameter = "EDRS"
        itemName = "EDR_ITEM"
        itemSeparator = "|"
        elementSeparator = ":"
        elementNames = [
            "RECORD_DATE",
            "WALLET_TYPE",
            "CHARGING_DOMAIN_ID",
            "CALL_ID",
            "SCP_ID",
            "SEQUENCE_NUMBER",
            "EXTRA_INFORMATION"
        ]
    }
    {
        command = "CCSVR1"
        action = "QRY"
        parameter = "BALANCES"
        itemName = "BALANCE_ITEM"
        itemSeparator = "|"
        elementSeparator = ":"
        elementNames = [
                    "BALANCE_TYPE",
                    "AMOUNT",
                    "POST_USE_EXPIRY",
                    "START_DATE",
                    "END_DATE",
                    "NEW_BUCKET",
                    "POLICY",
                    "MISSING_BALANCE_POLICY",
                    "REPLACE_BALANCE"
                ]
            }
        ]
```

eserv.config file example eserv.config.pi_example

PI comes with a file named **eserv.config.pi_example**. It is located in the root of the application directory. This file contains a commented example of the pi section of an **eserv.config** configuration file. As a starting point, when configuring features of the provisioning interface, copy **eserv.config.pi_example** into the main **eserv.config** file.

The content of the **eserv.config.pi_example** file is copied below. Most of the comments have been removed.

```
pi = {
    localTZ = "TimeZone"

    general = {
        # debug = 'N'
        # oraUser = "/"
        # synstamp = 'Y'
        # timeout = 30
        # logLevel = 0
        # securityPlugin = ""
        # correlationRequestTagName = "CORRELATE"
        # correlationResponseTagName = "CORRELATE"
    }

    throttling = {
        # sendRate = 0
    }

    PIbeClient = {

        clientName = "PIbeClient"

        oracleLogin = "/"

        heartbeatPeriod = 10000000

        maxOutstandingMessages = 100

        connectionRetryTime = 2

        plugins = [ # plugable functionality for the billing engine interface.
            { # Voucher recharge (VRW) plugin (need the broadcast plugin)
                config="voucherRechargeOptions",
                library="libccsClientPlugins.sl",
                function="makeVoucherRechargePlugin"
            }
            { # Broadcast plugin needed by VRW
                config="", # no config
                library="libclientBcast.sl",
                function="makeBroadcastPlugin"
            } # Broadcast one message to one BE of each pair
        # Activated by sending a message to BE ID 0.
    ]

    # Config for voucher recharge plugin
    voucherRechargeOptions = {
        # Should Scratch Card Alternate Subscriber activate a preuse a/c?
        # Not used by the PIbeClient
        srasActivatesPreuseAccount=false
        voucherRechargeTriggers = [
            "VRW " # this type of message triggers this plugin
        ]
    }

    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"}
    ]

    stateConversions = {
        A = "ACTV",
        P = "PREU",
        D = "DORM",
        F = "FROZ",
        S = "SUSP",
        T = "TERM"
    }

    voucherStateConversions = {
        A = "ACTV",
        F = "FRZN",
        R = "RDMD"
}

    # billingEngines = [
    #         {        id = 1, # pair ID
    #                  primary   = { ip="PRIMARY_BE_IP", port=1500 },
    #                  secondary = { ip="SECONDARY_BE_IP", port=1500 }
    #         }
    #]
}

ssl = {

    allowINSECURESSLv3 = false
    certificateFile = "/IN/service_packages/PI/my_sslCertificate.pem"
    keyFile = "/IN/service_packages/PI/my_sslKey.pem"
}

soap = {

    implicitLoginsSupported = false

    validateAuthStrings = true

    expansionRules = [

        {
            command = "CCSCD1"
            action = "QRY"
```

```
                parameter = "BALANCES"
                itemName = "BALANCE_ITEM"
                itemSeparator = "|"
                elementSeparator = ":"
                elementNames = [ "BALANCE_TYPE_NAME", "*BUCKETS" ]
            }
            {
                command = "CCSCD1"
                action = "QRY"
                parameter = "*BUCKETS"
                itemName = "BUCKET_ITEM"
                itemSeparator = "|"
                elementSeparator = ":"
                elementNames = [
                    "BUCKET_VALUE",
                    "BUCKET_EXPIRY"
                ]
            }
            {
                command = "CCSCD7"
                action = "QRY"
                parameter = "EDRS"
                itemName = "EDR_ITEM"
                itemSeparator = "|"
                elementSeparator = ":"
                elementNames = [
                    "RECORD_DATE",
                    "WALLET_TYPE",
                    "CHARGING_DOMAIN_ID",
                    "CALL_ID",
                    "SCP_ID",
                    "SEQUENCE_NUMBER",
                    "EXTRA_INFORMATION"
                ]
            }
            {
                command = "CCSVR1"
                action = "QRY"
                parameter = "BALANCES"
                itemName = "BALANCE_ITEM"
                itemSeparator = "|"
                elementSeparator = ":"
                elementNames = [
                    "BALANCE_TYPE",
                    "AMOUNT",
                    "POST_USE_EXPIRY",
                    "START_DATE",
                    "END_DATE",
                    "NEW_BUCKET",
                    "POLICY",
                    "MISSING_BALANCE_POLICY",
                    "REPLACE_BALANCE"
                ]
            }
        ]
    }
}
```

About Configuring PI Commands in eserv.config

Introduction

Some of the application commands support configuration in the **eserv.config** file.

Note: If the commands package which includes the related command have not be installed, these parameters are not supported.

Local time zone

The Local time zone subsection supports the following parameter.

```
    localTZ = "TimeZone"
```

The parameter is described in detail below.

localTZ

Syntax:	`localTZ = "TimeZone"`
Description:	Sets the time zone the PI uses for sending and receiving dates.
Type:	String
Optionality:	Optional (default used if not set).
Allowed:	The time zone name must be a valid UNIX time zone name such as CET or GMT.
Default:	GMT
Notes:	PI automatically accounts for summer time alterations within this time zone.
Example:	

Setting the Control Plan Export File Directory for ACSCPL PI Commands

You use the ACSCPL=EXP PI command to export control plans to **.cpl** files. The PI exports control plans to the following directory by default:

/IN/service_packages/PI/callplans

You can set a different control plan export directory by configuring the `exportCallPlanDirectory` parameter in the pi, ACSCPL section of the **eserv.config** configuration file:

```
pi = {
    ACSCPL = {
        exportCallPlanDirectory = "str"
        }
    }
```

The `exportCallPlanDirectory` parameter has the following characteristics:

exportCallPlanDirectory

Syntax:	`exportCallPlanDirectory = "str"`
Description:	The directory to which the PI exports control plan files.
Type:	String
Optionality:	Optional (default used if not set)
Default:	**/IN/service_packages/PI/callplans**
Example:	`exportCallPlanDirectory = "/IN/service_packages/PI/myControlPlans"`

Getting Information About Voucher Changes by Using PI Commands

When changing a voucher state or marking a voucher as frozen, you can use PI commands to record and return the reason for the change, and the user to whom it applied.

To record the data, the CSVR1=CHG command (change voucher status) and CCSVR1=FRZ command (mark voucher frozen) have an optional DESCRIPTION parameter that you can use to record the reason for the action.

To retrieve the data, the CCSVR1=QRY command (query a recharge voucher) can return the following information:

- The reason for a voucher state change. This information is in the DESCRIPTION field. This field is limited to 50 characters and is truncated if the input is too long.
- The user for the voucher state change. This information is in the STATE_CHANGE_USER field.

You can disable having the CCSVR1=QRY command return the DESCRIPTION and STATE_CHANGE_USER fields by editing the `pi.CCSVR1.QRY.suppressField` in the **eserv.config** file. To configure this entry, enter the fields you want to suppress, separated by the pipe (**|**) character. The default is to display all fields. This entry is read only on the first call to CCSVR1=QRY.

The following example suppresses both fields; DESCRIPTION and STATE_CHANGE_USER:

```
pi = {
    CCSVR1 = {
        QRY = {
            suppressFields = "DESCRIPTION|STATE_CHANGE_USER"
        }
    }
}
```

CCSCD1

The `CCSCD1` subsection of the PI **eserv.config** configuration supports these parameters.

```
    CCSCD1 = {
        ADD = {
            initialState = "state"
            noWalletCreateBeIds = [ BE1, BE2 ]
            useSystemLanguage = 'Y|N'

        }
        CHG = {
            createEmptyBalance = true|false
        }
        QRY = {
            currencyType = "str"
        }
    }
```

The parameters in the ADD, CHG, and QRY subsections are described in detail below.

initialState

Syntax:	`initialState = "`*state*`"`
Description:	The initial wallet state for wallets created using the CCSCD1=ADD_INITIAL_STATE command.
Type:	String
Optionality:	Optional

Allowed:	P	Pre-use
	A	Active
	D	Dormant
	S	Suspended
	F	Frozen
	T	Terminated

Default: P

Notes: For more information about the CCSCD1 command, see *CCS Provisioning Interface Commands*.

Example: `initialState = "P"`

noWalletCreateBeIds

Syntax: `noWalletCreateBeIds = [ be_ids ]`

Description: Comma separated list of billing engine IDs that may not be used for wallet creation.

Type: Array

Optionality: Optional (default used if missing).

Allowed: List of valid billing engine IDs.

Default: Not set

Notes:

Example: `noWalletCreateBeIds = [ 1, 2 ]`

createEmptyBalance

Syntax: `createEmptyBalance = true|false`

Description: When `createEmptyBalance` is set to true, allows a balance with no existing buckets to be created with a zero (0.0) value.

Type: Boolean

Optionality: Optional (default used if not set)

Allowed:

- true – Allow balances to be created with a zero value.
- false – Do not allow balances to be created with a zero value.

Default: false

Notes: Setting `createEmptyBalance` to true enables you to use the CCSCD1=CHG PI command to provision the expiry date for the balance so that any subsequent recharges into the balance have the correct expiry date extension (based on the "Best" balance expiry policy).

Example: `createEmptyBalance = true`

currencyType

Syntax: `currencyType = "str"`

Description: Sets the type of currency.

Type: String

Optionality: Optional

Allowed:	user	Use the user's wallet currency type.
	system	Use the system currency type.

Default: user

Notes:	For more information about the CCSCD1 command, see *CCS Provisioning Interface Commands*.
Example:	`currencyType = "user"`

useSystemLanguage

Syntax:	`useSystemLanguage = 'Y|N'`
Description:	Sets whether to use the system language for new subscribers, or the subscriber's language.
Type:	Boolean
Optionality:	Optional (default used if not set)
Allowed:	Y – Use the system language N – Use the subscriber's language
Default:	`Y`
Notes:	
Example:	`useSystemLanguage = 'N'`

CCSCD3

The `CCSCD3` subsection of the PI **eserv.config** configuration supports these parameters.

```
CCSCD3 = {
    CTR = {
        creditTransferCP = "creditTransferControlPlan"
    }
    RCH = {
        fixedVoucherNumberLength = 10
        defaultScenarioName = "str"
        activatePreuseAccount = "true"
    }
}
```

The parameters in this subsection are described in detail below.

creditTransferCP

The name of the credit transfer control plan.

Default:	`"CREDIT_TRANSFER"`
Type:	String

fixedVoucherNumberLength

Syntax:	`fixedVoucherNumberLength = num`
Description:	The voucher number length. Must be set correctly when performing scenario recharges and the scenario name is specified.
Type:	Integer
Optionality:	Optional (default used if not set).
Allowed:	Valid voucher number length
Default:	10
Notes:	
Example:	`fixedVoucherNumberLength = 10`

`defaultScenarioName`

Syntax:	`defaultScenarioName = "`*`scenario`*`"`
Description:	Specifies the default scenario to use.
Type:	String
Optionality:	Optional (default used if not set).
Allowed:	A valid scenario name.
Default:	Default
Notes:	
Example:	`defaultScenarioName = "Default"`

`activatePreuseAccount`

Syntax:	`activatePreuseAccount = "true|false"`
Description:	Sets whether or not to activate pre-use wallets for recharge attempts.
Type:	Boolean
Optionality:	Optional (default used if not set).
Allowed:	• true – Activate pre-use wallets for recharge attempts. • false – Do not activate pre-use wallets.
Default:	true
Notes:	Quoted value is required.
Example:	`activatePreuseAccount = "false"`

CCSBPL

The `CCSBPL` subsection of the PI **eserv.config** configuration supports these parameters.

```
CCSBPL = {
    notifyEagain = true
    maxFifoReadRetry = 10
    triggerTimeoutSeconds = 10
    }
```

The parameters in this subsection are described in detail below.

`notifyEagain`

Syntax:	`notifyEagain = true|false`
Description:	Whether or not to display notice alarms for missed reads from the Fifo queue.
Type:	Boolean
Optionality:	Optional (default used if not set).
Allowed:	• true – Display notice alarms for missed reads. • false – Do not display notice alarms for missed reads.
Default:	false
Notes:	
Example:	`notifyEagain = true`

`maxFifoReadRetry`

Syntax:	`maxFifoReadRetry = `*`maximum`*
Description:	Sets the maximum number of times to retry reading from the Fifo queue.
Type:	Integer

Optionality:	Optional (default used if not set).
Allowed:	
Default:	10
Notes:	
Example:	`maxFifoReadRetry = 20`

triggerTimeoutSeconds

Syntax:	`triggerTimeoutSeconds =` *`seconds`*
Description:	Sets the timeout, in seconds, for waiting for a response from smsTrigDaemon.
Type:	Integer
Optionality:	Optional (default used if not set)
Allowed:	
Default:	10
Notes:	
Example:	`triggerTimeoutSeconds = 5`

CCSVR1

The `CCSVR1` subsection of the PI **eserv.config** configuration supports these parameters.

```
CCSVR1 = {
    acsCustomerId = 0|1
    QRY= {
        suppressScenario = 'Y|N'
        suppressFields = "str1|str2"
        DECRYPT_PRIVATE_SECRET = 'Y|N'
    }
    COV = {
         DEFAULT_PROVIDER = “provider_name”
         DEFAULT_VOUCHER_TYPE = “voucher_type_name”
         DEFAULT_AUTH_MODULE_NAME = “auth_module_name”
         DEFAULT_AUTH_RULE_NAME = “auth_rule_name”
         DEFAULT_INITIAL_STATE = “A|C|F”
         DEFAULT_BATCH_SIZE = “number”
         RETRY_COUNT = “number”
         RETRY_INTERVAL = “seconds”
    }
}
```

acsCustomerId

Syntax:	`acsCustomerId = nnn` where nnn is a positive number
Description:	Sets the default acsCustomerId to use for CCSVR1 commands where PROVIDER is not specified on the command.
Type:	Integer
Optionality:	Optional. If not defined, no default is used and PROVIDER must be specified on the CCSVR1 command.
Allowed:	nnn Must be an existing numeric service provider Id (> 0).
Default:	No default
Notes:	Sending PROVIDER in the PI command line overrides this value.
Example:	`acsCustomerId = 1`

suppressScenario

Syntax:	`suppressScenario = 'Y|N'`
Description:	Determines whether or not to suppress the SCENARIO return parameter if the voucher has an associated scenario.
Type:	Boolean
Optionality:	Optional (default used if not set).
Allowed:	Y — Do not return the SCENARIO return parameter even if the voucher has an associated scenario. N — Return the SCENARIO return parameter if the voucher has an associated scenario.
Default:	N
Notes:	If an invalid parameter value is specified, then 'N' is assumed and an error alarm is output to the PImanager log file.
Example:	`suppressScenario = 'N'`

suppressFields

Syntax:	`suppressFields = "str1|str2.."`
Description:	Lists the fields to suppress from the results displayed for the CCSVR1=QRY PI command, where *str1* and *str2* are fields output by the CCSVR1 query command.
Type:	Parameter list
Optionality:	Optional (default used if not set)
Allowed:	
Default:	Display all fields
Notes:	
Example:	`suppressFields = "DESCRIPTION|STATE_CHANGE_USER"`

decrypt_private_secret

Syntax:	`DECRYPT_PRIVATE_SECRET = value`
Description:	The voucher private secret will be decrypted to obtain the original HRN.
Type:	Boolean
Optionality:	Optional (default used if not set)
Allowed:	Y or N
Default:	N
Notes:	
Example:	`DECRYPT_PRIVATE_SECRET = Y`

DEFAULT_PROVIDER

Syntax:	`DEFAULT_PROVIDER = "provider_name"`
Description:	Default provider name to be used for CCSVR1=COV, when not provided in the request.
Type:	string
Optionality:	Optional. If not defined, no default is used and PROVIDER must be specified on the CCSVR1=COV command.

Allowed:	Must be an existing service provider name.
Default:	No default
Notes:	Sending PROVIDER in the PI command line overrides this value.
Example:	`DEFAULT_PROVIDER = "OCNCCtemplate"`

DEFAULT_VOUCHER_TYPE

Syntax:	`DEFAULT_VOUCHER_TYPE = "`*`voucher_type_name`*`"`
Description:	Default voucher type to be used for CCSVR1=COV, when not provided in the request.
Type:	string
Optionality:	Optional. If not defined, no default is used and VOUCHER_TYPE must be specified on the CCSVR1=COV command.
Allowed:	Must be an existing voucher type name.
Default:	No default
Notes:	Sending VOUCHER_TYPE in the PI command line overrides this value.
Example:	`DEFAULT_VOUCHER_TYPE = "VT1"`

DEFAULT_AUTH_MODULE_NAME

Syntax:	`DEFAULT_AUTH_MODULE_NAME = "`*`auth_module_name`*`"`
Description:	Default authentication module name to be used for CCSVR1=COV, when not provided in request.
Type:	string
Optionality:	Optional. If not defined, no default is used and AUTH_MODULE_NAME must be specified on the CCSVR1=COV command.
Allowed:	Must be an existing authentication module name.
Default:	No default
Notes:	Sending AUTH_MODULE_NAME in the PI command line overrides this value.
Example:	`DEFAULT_AUTH_MODULE_NAME = "CB10 HRN SHA512"`

DEFAULT_AUTH_RULE_NAME

Syntax:	`DEFAULT_AUTH_RULE_NAME = "`*`auth_rule_name`*`"`
Description:	Default authentication rule name to be used for CCSVR1=COV, when not provided in request.
Type:	string
Optionality:	Optional. If not defined, no default is used and AUTH_RULE_NAME must be specified on the CCSVR1=COV command.
Allowed:	Must be an existing authentication rule name. Authentication rule provided here must be defined under the authentication module name configured with DEFAULT_AUTH_MODULE_NAME parameter.
Default:	No default
Notes:	Sending AUTH_RULE_NAME in the PI command line overrides this value.
Example:	`DEFAULT_AUTH_RULE_NAME = "SHA512"`

DEFAULT_INITIAL_STATE

Syntax:	`DEFAULT_INITIAL_STATE = "A | C | F"`
Description:	Default value for initial state of voucher (when created) to be used for CCSVR1=COV, when not provided in request.

Type:	string
Optionality:	Optional. If not defined, no default is used and INITIAL_STATE must be specified on the CCSVR1=COV command.
Allowed:	**C** Generate vouchers with state = Created
	A Generate vouchers with state = Activated
	F Generate vouchers with state = Frozen
Default:	No default
Notes:	Sending INITIAL_STATE in the PI command line overrides this value.
Example:	`DEFAULT_INITIAL_STATE = "A"`

DEFAULT_BATCH_SIZE

Syntax:	`DEFAULT_BATCH_SIZE = "`*`batch_size`*`"`
Description:	Default value for 'On Demand Voucher' batch size to be used for CCSVR1=COV.
Type:	string
Optionality:	Optional. If not defined, default will be used.
Allowed:	Positive Number
Default:	10000
Notes:	On demand batch is created, when there is none available, or if the available ones are full in capacity. Keeping a lower value for this parameter will cause batches to be created too frequently. It is recommended to keep a higher value for this parameter depending on the rate of 'On Demand Voucher' creations.
Example:	`DEFAULT_BATCH_SIZE = "10000"`

RETRY_COUNT

Syntax:	`RETRY_COUNT = "`*`count`*`"`
Description:	Number of retries to be done for executing the ccsVoucher binary during a request for CCSVR1=COV.
Type:	string
Optionality:	Optional. If not defined, default will be used.
Allowed:	Positive Number
Default:	3
Notes:	
Example:	`RETRY_COUNT = "3"`

RETRY_INTERVAL

Syntax:	`RETRY_INTERVAL = "`*`seconds`*`"`
Description:	Number of seconds to wait before retrying execution of the ccsVoucher binary during a request for CCSVR1=COV.
Type:	string
Optionality:	Optional. If not defined, default will be used.
Allowed:	Positive Number
Default:	5
Notes:	
Example:	`RETRY_INTERVAL = "5"`

DEFAULT_DOMAIN_NAME

Syntax:	`DEFAULT_DOMAIN_NAME = "`*`billingEngineName`*`"`
Description:	The name of the domain or billing engine under which on-demand voucher needs to be generated. This default value is used when the CCSVR1=COV request doesn't contain DOMAIN_NAME.
Type:	string
Optionality:	Optional.
Allowed:	The domain name must exist in the database.
Default:	No default
Notes:	Co-relation between request parameter DOMAIN_NAME and **eserv.config** parameter DEFAULT_DOMAIN_NAME. • DOMAIN_NAME has precedence over DEFAULT_DOMAIN_NAME. • If DOMAIN_NAME is supplied in CCSVR1=COV request, then DEFAULT_DOMAIN_NAME is not used. • If DOMAIN_NAME is not supplied in CCSVR1=COV request, then DEFAULT_DOMAIN_NAME is used. • If DOMAIN_NAME is not supplied in CCSVR1=COV request, and the DEFAULT_DOMAIN_NAME is not specified in **eserv.config**, then: ▪ If there is only one domain in set-up: The only available domain is used in voucher generation. ▪ If there are multiple domains in set-up: This is an error scenario. You must specify domain either in request, or define a default value in **eserv.config** file.
Example:	`DEFAULT_DOMAIN_NAME="VWS_DOMAIN"`

CCSCD9

The `CCSCD9` subsection of the PI **eserv.config** configuration supports these parameters.

```
CCSCD9 = {
    QRY= {
        suppressEmptyField = true|false
    }
}
```

The parameter in the QRY subsection is described in detail below.

suppressEmptyFields

Syntax:	`suppressEmptyFields = true|false`
Description:	Specifies how to handle empty profile tag name fields.
Type:	Boolean
Optionality:	Optional (default used if not set)
Allowed:	• true – If the tag name field is not in the profile, no tag name is printed. • false – It prints empty profile tag name fields with a null value.
Default:	true
Notes:	
Example:	`suppressEmptyFields = true`

CCSSC1

`defaultBEDomainID`

Syntax:	`defaultBEDomainID = `*`id`*
Description:	BE pair ID to query when no SUBSCRIBER supplied to query for their BE pair supplied for CCSSC1=QRY command.
Type:	Integer
Optionality:	Optional (default used if not set).
Allowed:	
Default:	-1
Notes:	Must match a value in the CCS_DOMAIN.DOMAIN_ID database table. -1 = find the first domain that supports charging.
Example:	`defaultBEDomainID = 2`

Specifying the Maximum PQYZ Records to Query in the NP Database

You use the NPYZ1=QRY PI command to query the NP database for PQYZ entries. By default, the maximum number of records returned is 1500. You can specify a different maximum by configuring the `pqyzMaxRecords` parameter in the pi, NP section of the **eserv.config** configuration file:

```
pi = {
    NP = {
        pqyzMaxRecords = int
    }
}
```

The `pqyzMaxRecords` parameter has the following characteristics:

`pqyzMaxRecords`

Syntax:	`pqyzMaxRecords = `*`int`*
Description:	The maximum number of records returned when you query the NP database for multiple PQYZ entries that match one or more destination addresses.
Type:	Integer
Optionality:	Optional (default used if not set)
Default:	1500
Notes:	The PI outputs an error if the query finds more records than the configured maximum.
Example:	`pqyzMaxRecords = 500`

Defining the Screen Language

Introduction

The default language file sets the language that the Java administration screens start in. The user can change to another language after logging in.

The default language can be changed by the system administrator.

By default, the language is set to English. If English is your preferred language, you can skip this step and proceed to the next configuration task, *Defining the Help Screen Language* (on page 42).

Default.lang

When PI is installed, a file called `Default.lang` is created in the application's language directory in the screens module. This contains a soft-link to the language file which defines the language which will be used by the screens.

If a **Default.lang** file is not present, the **English.lang** file will be used.

The PI **Default.lang** file is **/IN/html/PI/language/Default.lang**.

Example Screen Language

If Dutch is the language you want to set as the default, create a soft-link from the **Default.lang** file to the **Dutch.lang** file.

Procedure

Follow these steps to set the default language for your PI Java administration screens.

Step	Action
1	Change to the following directory: `/IN/html/PI/language` **Example command:** `cd /IN/html/PI/language`
2	Check that the **Default.lang** file exists in this directory.
3	If the required file does not exist, create an empty file called **Default.lang**.

4	Ensure that the language file for your language exists in this directory. The file should be in the format:
	`language.lang`
	Where:
	language = your language.
	Example:
	`Spanish.lang`

5	If the required language file does not exist, perform one of the following actions: • Create a new one with your language preferences • Contact Oracle support To create a language file, you will need a list of the phrases and words used in the screens. These should appear in a list with the translated phrase in the following format: `original phrase=translated phrase` Any existing language file should have the full set of phrases. If you do not have an existing file to work from, contact Oracle support with details.
6	Create a soft link between the **Default.lang** file, and the language file you want to use as the default language for the PI Java administration screens. **Example command:** `ln -s Dutch.lang Default.lang`

Defining the Help Screen Language

Introduction

The default Helpset file sets the language that the help system for the Java Administration screens start in. The user can change to another language after logging in.

The default language can be changed by the system administrator. By default, the language is set to English.

Default_PI.hs

When PI is installed, a file called **Default_PI.hs** is created in the application's language directory in the screens module. This contains a soft-link to the language file which defines the language which will be used by the screens.

If a **Default_PI.hs** file is not present, the **English_PI.hs** file will be used.

If a **Default_PI.hs** file is present, the default language will be used.

The default file is **/IN/html/PI/helptext/Default_PI.hs**.

Example helpset language

If Dutch is the language you want to set as the default, create a soft-link from the **Default_PI.hs** file to the **Dutch_PI.hs** file.

Procedure

Follow these steps to set the default language for your PI Java Administration screens.

Step	Action
1	Change to the following directory: `/IN/html/PI/helptext` **Example command:** `cd /IN/html/PI/helptext`
2	check that the **Default_PI.hs** file exists in this directory.
3	If the required file does not exist, create an empty file called **Default_PI.hs**.
4	Ensure that the language file for your language exists in this directory. The file should be in the format: *`language`*`_PI.hs` **Where:** *language* = your language. **Example:** Dutch_PI.hs
5	If the required language file does not exist, perform one of the following: • Create a new one with your language preferences • Contact Oracle support To create a language file, you will need a list of the phrases and words used in the screens. These should appear in a list with the translated phrase in the following format: `original phrase=translated phrase` Any existing language file should have the full set of phrases. If you do not have an existing file to work from, contact Oracle support with details.
6	Create a soft link between the **Default_PI.hs** file, and the language file you want to use as the default language for the SMS Java administration screens. **Example command:** `ln -s Dutch_PI.hs Default_PI.hs`

Defining a Password Policy

Password verification functions are a feature in the Oracle database and allow a site to configure their own, usually stricter, password verification policy. A password policy is site dependent, based on the password verification function specified.

The strength checking function for a PI user password is listed in a database profile called PI_USERS. This profile is used globally for all the PI users.

The profile also contains limits:

- PASSWORD_LIFE_TIME: The number of days after a password change that a password is unusable. After this time, at PI login, the validation fails and the password must be reset. If the PI password has expired, then a cmnError ERROR message is logged at login time.
- PASSWORD_GRACE_TIME: The number of days before a password expires that login is still allowed with a cmnError WARNING message being logged that the password is close to expiry.

When a new PI user is created, and or an existing PI user's password is changed, their password strength is checked. If this check fails, an error message is displayed. This error message is configurable and is set by a parameter in **sms.jnlp**.

The following example shows the parameter in **sms.jnlp**:

```
<property name="jnlp.sms.piUsersPasswordPolicyMessage" value="The new password must
be at least 9 characters long and have at least 2 uppercase characters, 2 lowercase
characters, 2 digits and 2 special characters, and must be 4 characters or more
different from the previous password if there was one." />
```

Note: This policy message describes the policy provided by NCC by default for PI Users, which uses the ORA12C_STRONG_VERIFY_FUNCTION to verify the strength of PI User passwords.

If this parameter is not set, then a default message saying the password has failed the PI password policy is displayed.

When a user logs into PI, if the verification of the user password fails, then the number of the failed login attempts is incremented. If this value reaches 3, then the PI user's account is locked and the Lock Reason field in the PI Users tab displays the reason. To continue using the account, the account must then be unlocked by changing the password or clearing the Lock Reason field for the PI User in the PI Users tab. When a password is reset, the Failed Logins field is set to 0, the Lock Reason is cleared, and the Last Password Change field is set to the current time.

Password Policy Requirements

A password policy allows a site to specify that a password meets a number of requirements. For example, the requirements could include one or more of the following requirements:

- Contains a minimum number of characters
- Contains a minimum number of digits
- Contains a minimum number of letters
- Contains a minimum number of special characters
- Does not contain double-quote characters
- Differs from a previous password by a specified number of characters

Example Password Policy

The following is an example of a password policy:

- Contains a minimum of 9 characters
- Contains a minimum of 2 digits
- Contains a minimum of 2 letters
- Contains a minimum of 2 special characters
- Differs from a previous password by 4 characters

Password Verification Function

The PI_USERS database profile uses the function named by resource_name PASSWORD_VERIFY_FUNCTION as the name of the database function to call for PI User password strength checking. By default, PASSWORD_VERIFY_FUNCTION is set to ORA12C_STRONG_VERIFY_FUNCTION, which is a verify function supplied with Oracle 12c.

It is possible to change the password verify function to another one.

The following rules should be followed for any new password verification function:

1 It must be a PL/SQL function with the following signature:
`routine_name` (userid_parameter IN VARCHAR2, password_parameter IN VARCHAR2, old_password_parameter IN VARCHAR2) RETURN BOOLEAN

2 The function must be owned by the SYS user.

3 The function must have execute permission for PUBLIC granted to it.

The database-provided SQL file ($ORACLE_HOME/rdbms/admin/utlpwdmg.sql) defines the standard password verification functions and additional helper functions.

Chapter 3

PI Administration Screen

Overview

Introduction

This chapter explains how to use the PI Administration screen.

In this chapter

This chapter contains the following topics.

PI Administration Screen

Introduction

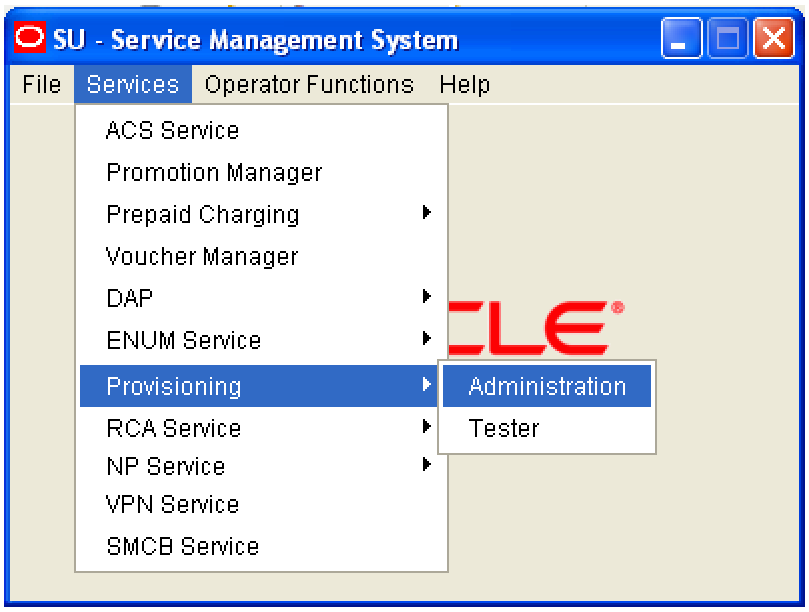
The Administration screen manages users and templates in the Provisioning Interface. It contains these tabs:

- Commands
- Hosts
- MAC Pairs
- Users
- Ports
- Password Expiry

Accessing the PI Administration screen

Follow the steps below to access the Administration screen.

Step	Action
1	Select the **Services** menu from the Service Management System main menu.

Step	Action
	
2	Select **Provisioning**.
3	Select **Administration**.
	Result: You see the PI Administration screen, showing the **Commands** tab.

PI Commands

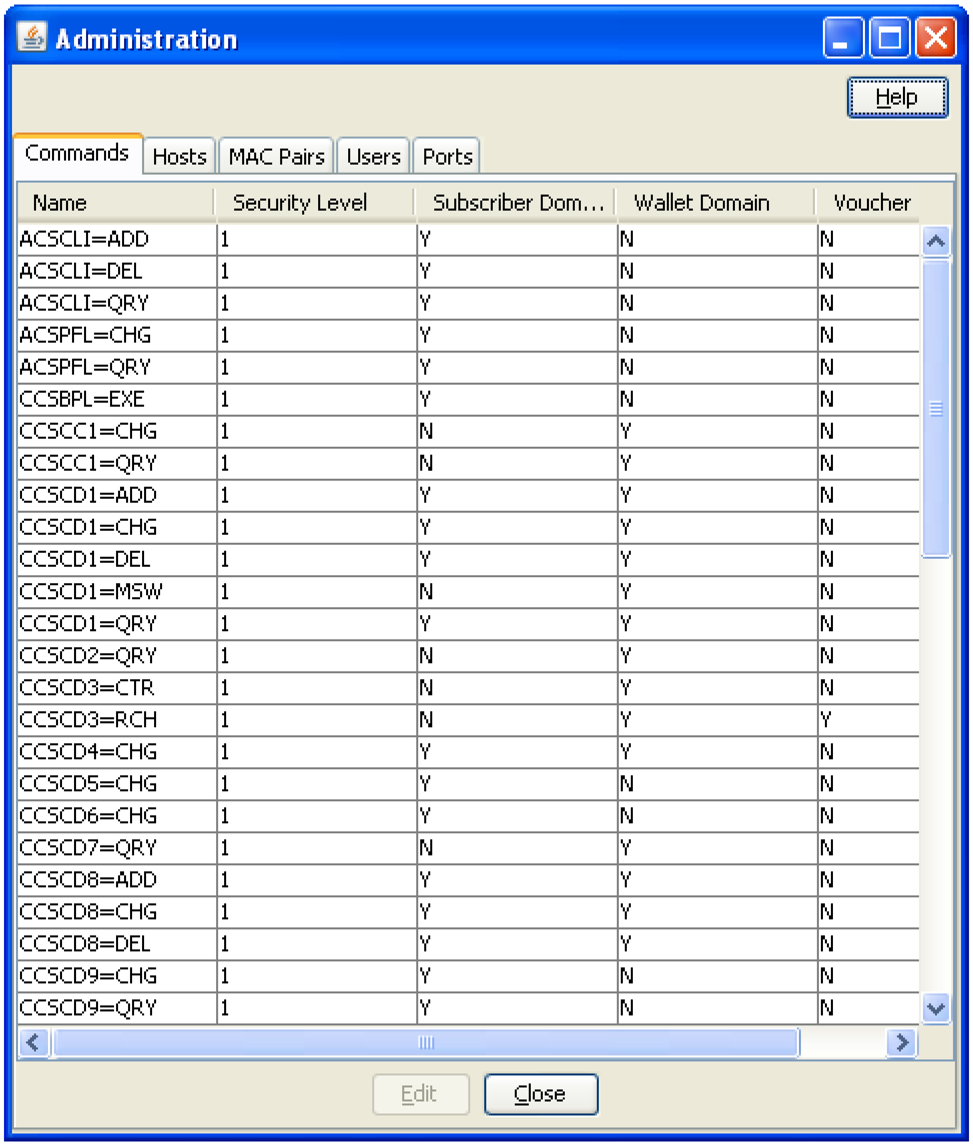
Introduction

The **Commands** tab of the PI Administration screen enables you to set the security level for PI commands.

Note: Commands cannot be added to or removed from the list of available commands.

Commands tab

Here is an example **Commands** tab.



Name	Security Level	Subscriber Dom...	Wallet Domain	Voucher
ACSCLI=ADD	1	Y	N	N
ACSCLI=DEL	1	Y	N	N
ACSCLI=QRY	1	Y	N	N
ACSPFL=CHG	1	Y	N	N
ACSPFL=QRY	1	Y	N	N
CCSBPL=EXE	1	Y	N	N
CCSCC1=CHG	1	N	Y	N
CCSCC1=QRY	1	N	Y	N
CCSCD1=ADD	1	Y	Y	N
CCSCD1=CHG	1	Y	Y	N
CCSCD1=DEL	1	Y	Y	N
CCSCD1=MSW	1	N	Y	N
CCSCD1=QRY	1	Y	Y	N
CCSCD2=QRY	1	N	Y	N
CCSCD3=CTR	1	N	Y	N
CCSCD3=RCH	1	N	Y	Y
CCSCD4=CHG	1	Y	Y	N
CCSCD5=CHG	1	Y	N	N
CCSCD6=CHG	1	Y	N	N
CCSCD7=QRY	1	N	Y	N
CCSCD8=ADD	1	Y	Y	N
CCSCD8=CHG	1	Y	Y	N
CCSCD8=DEL	1	Y	Y	N
CCSCD9=CHG	1	Y	N	N
CCSCD9=QRY	1	Y	N	N

Commands fields

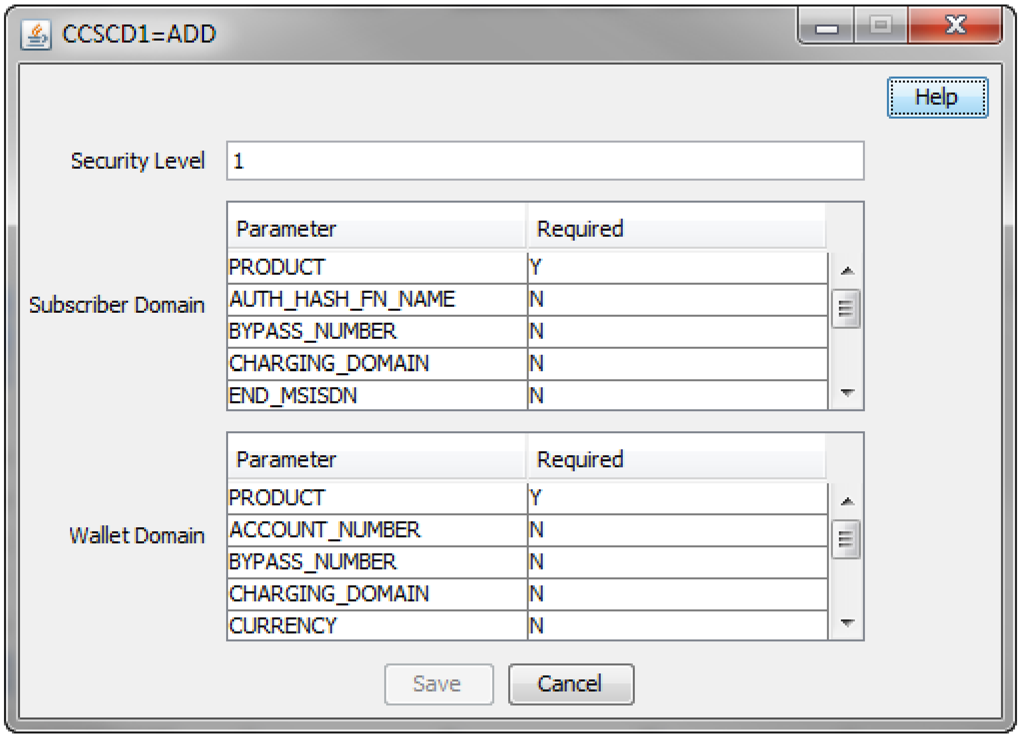
This table describes the function of each field.

Field	Description
Name	The PI command name.
Security Level	The security level required to execute the command.
Subscriber Domain	Indicates the command applies to an account that belongs to the subscriber domain, that is, the account exists on the SMS, the account wallet may be on a VWS or on a third party billing engine.
Wallet Domain	Indicates the command applies to an account that belongs to the wallet domain, that is, both the account and wallet exists on the SMS and VWS.

Field	Description
Voucher Domain	Indicates the command applies to a voucher that belongs to the voucher domain, that is, a voucher that exists on the VWS.

Example screen

The following example shows the edit dialog box for the CCSCD1=ADD PI command.



CCSCD1=ADD

Help

Security Level: 1

Subscriber Domain

Parameter	Required
PRODUCT	Y
AUTH_HASH_FN_NAME	N
BYPASS_NUMBER	N
CHARGING_DOMAIN	N
END_MSISDN	N

Wallet Domain

Parameter	Required
PRODUCT	Y
ACCOUNT_NUMBER	N
BYPASS_NUMBER	N
CHARGING_DOMAIN	N
CURRENCY	N

Editing PI commands

Follow these steps to edit a PI command.

Step	Action
1	From the list of PI commands on the **Commands** tab, select the command you want to edit.
2	Click **Edit**. **Result:** The edit dialog box for the selected command appears. See *Commands fields* (on page 49) for a description of each field.
3	Change the **Security Level** as required. **Note:** Range is 1 to 99 (highest) inclusive.
4	Click **Save**. **Result:** The details are saved to the database.
5	Soft restart the PI. For details, see *Soft PI Restart* (on page 76). **Result:** The updated configuration details will be loaded by the PImanager.

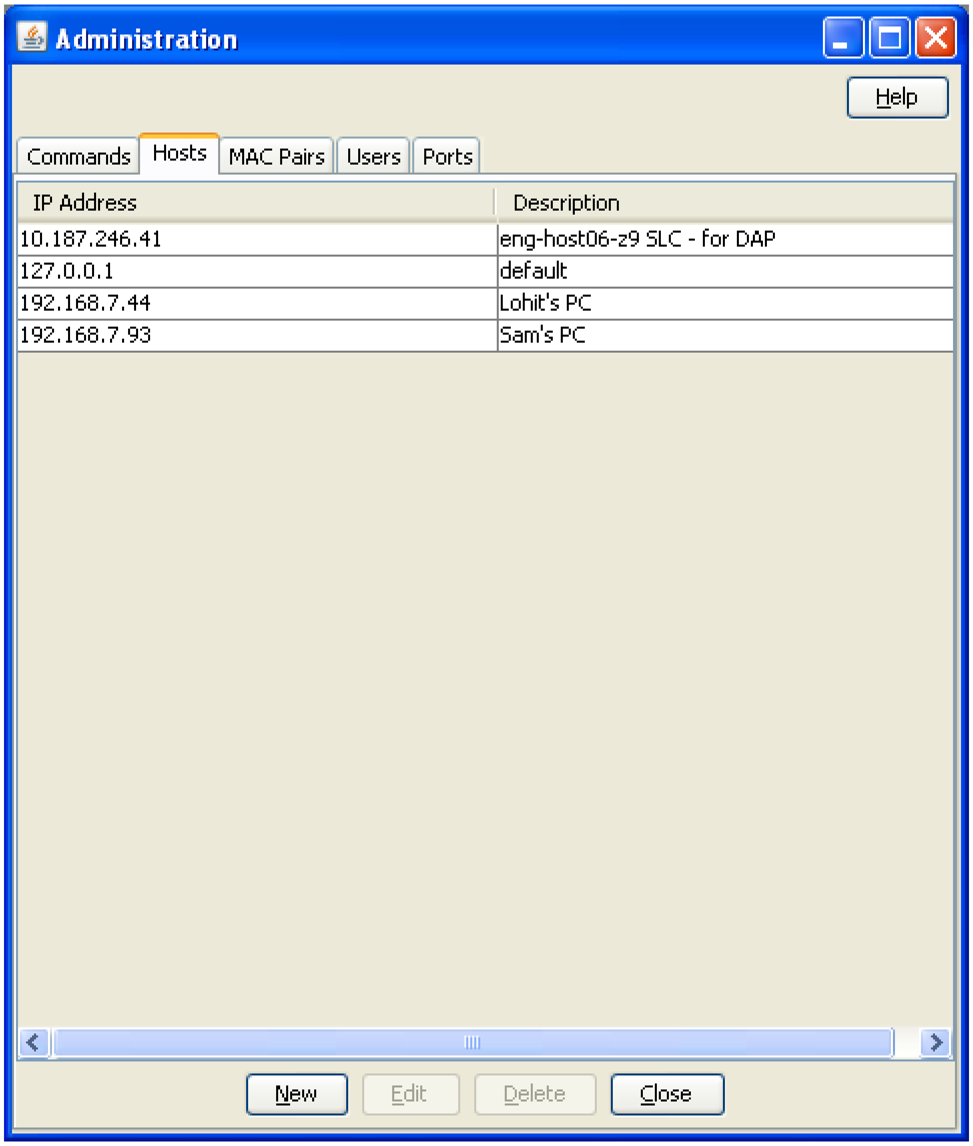
PI Hosts

Introduction

The **Hosts** tab of the PI Administration screen enables you to configure the hosts from which PI commands can be run. Before a new client can connect, it must be added to the database. You can also use IP class. The list of clients whose IP address belongs to that subnet class can also connect. For example, 198.51.100.250/24.

Hosts tab

Here is an example **Hosts** tab.



IP Address	Description
10.187.246.41	eng-host06-z9 SLC - for DAP
127.0.0.1	default
192.168.7.44	Lohit's PC
192.168.7.93	Sam's PC

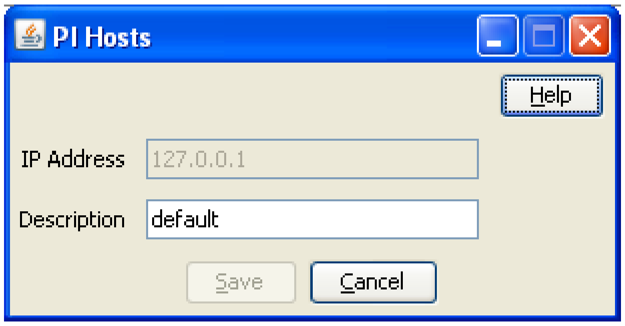
Hosts fields

This table describes the function of each field.

Field	Description
IP Address	The unique IP address of the host which will be allowed to run commands in the PI. IP class is also supported. For example, 198.51.100.250/24. **Note:** You cannot modify the IP address after it is first saved.
Description	A description of the host defined in the **IP Address** field, such as the hostname. The PI does not use the description value when connecting to the host.

PI Hosts screen

Here is an example PI Hosts screen.



IP Address 127.0.0.1
Description default

Adding hosts

Follow these steps to add new hosts to the PI.

Step	Action
1	On the **Hosts** tab, click **New**. **Result:** The *PI Hosts screen* (on page 52) displays. See *Hosts fields* (on page 52) for a description of each field.
2	In the **IP Address** field, type the IP address of the host.
3	In the **Description** field, type a description for the host, such as the hostname.
4	Click **Save**. **Result:** The new host details are saved in the database.
5	Soft restart the PI. For details, see *Soft PI Restart* (on page 76). **Result:** The updated configuration details will be loaded by the PImanager.

Editing hosts

Follow these steps to edit host information in the PI.

Step	Action
1	On the **Hosts** tab, select from the list the host to edit.
2	Click **Edit**. **Result:** The PI Hosts screen appears showing the data for the selected host record. See *Hosts fields* (on page 52) for a description of each field.
3	Change the host **Description** as required.

Step	Action
4	Click **Save**.
	Result: The details are saved to the database.
5	Soft restart the PI. For details, see *Soft PI Restart* (on page 76).
	Result: The updated configuration details will be loaded by the PImanager.

Deleting hosts

Follow these steps to delete a host from the PI.

Step	Action
1	In the **Hosts** tab, select from the list the host to delete.
2	Click **Delete**.
	Result: The Delete Confirmation screen displays.
3	Click **OK**.
	Result: The host is removed from the database.
4	Soft restart the PI. For details, see *Soft PI Restart* (on page 76).
	Result: The updated configuration details will be loaded by the PImanager.

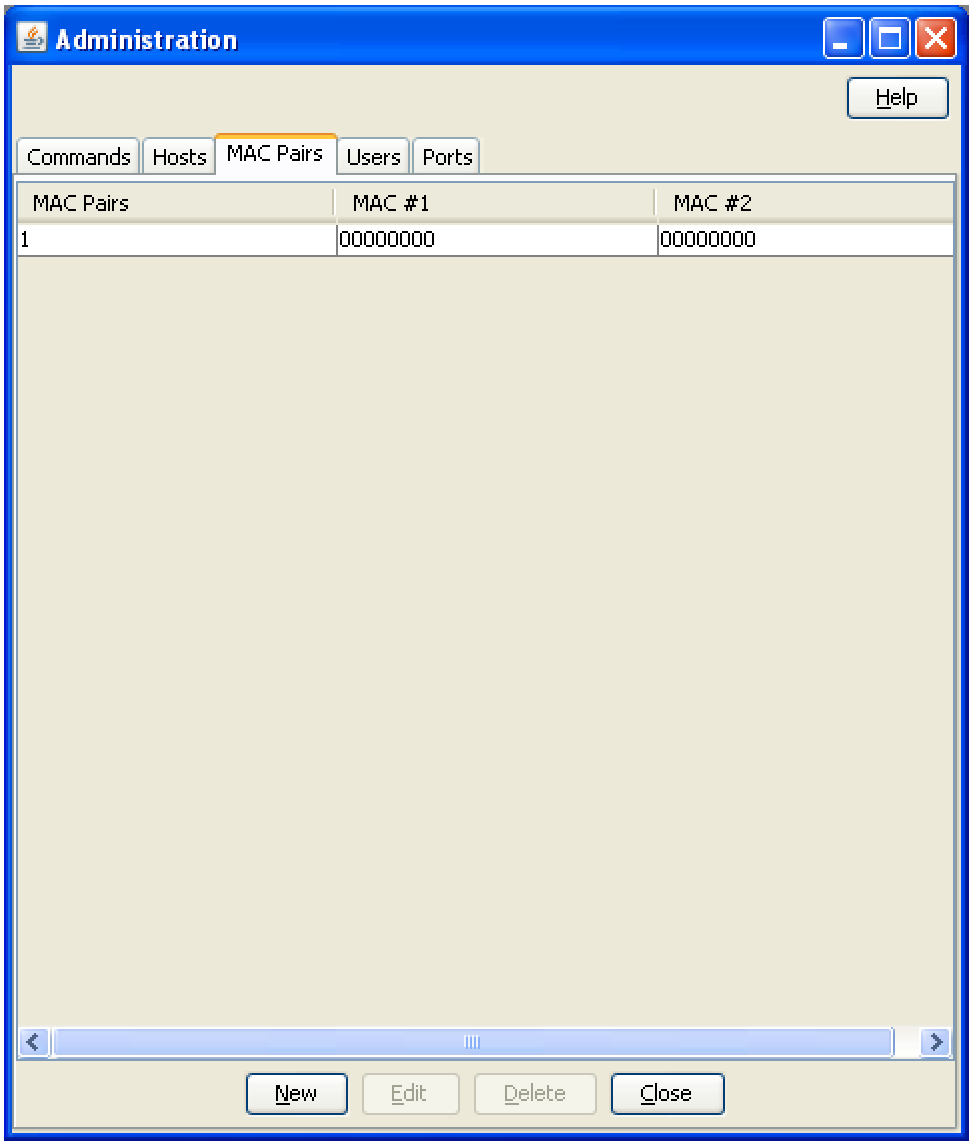
PI MAC Pairs

Introduction

The **MAC Pairs** tab of the Administration screen enables you to configure the MAC pairs from which commands can be run in PI. MAC pairs are the security keys to encode and decode encrypted data.

MAC Pairs tab

Here is an example **MAC Pairs** tab.



MAC Pairs	MAC #1	MAC #2
1	00000000	00000000

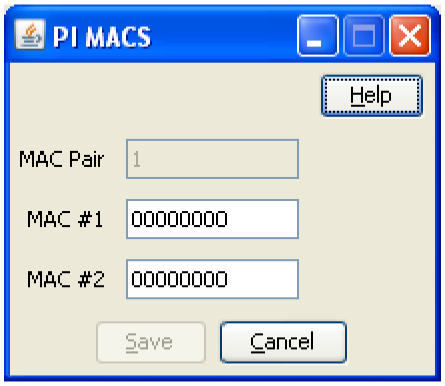
MAC Pairs fields

This table describes the function of each field.

Field	**Description**
MAC Pair	The unique MAC pair number for this MAC pair. **Note:** This field cannot be changed after it is first saved.
MAC #1	The MAC address of the first MAC address in this MAC pair. This must be an 8 digit number.
MAC #2	The MAC address of the second MAC address in this MAC pair. This must be an 8 digit number.

PI MACS screen

Here is an example PI MACS edit screen.



MAC Pair: 1
MAC #1: 00000000
MAC #2: 00000000

Adding MAC Pairs

Follow these steps to add new MAC pairs to the PI.

Step	Action
1	On the **MAC Pairs** tab, click **New**. **Result:** The *PI MACS screen* (on page 55) displays. See *MAC Pairs fields* (on page 54) for a description of each field.
2	Enter in the **MAC Pair** field the unique MAC pair number.
3	Enter in the **MAC #1** field the MAC address of the first entry for the MAC pair.
4	Enter in the **MAC #2** field the MAC address of the second entry for the MAC pair.
5	Click **Save**. **Result:** The new MAC pair details are saved in the database.
6	Soft restart the PI. For details, see *Soft PI Restart* (on page 76). **Result:** The updated configuration details will be loaded by the PImanager.

Editing MAC Pairs

Follow these steps to edit MAC pair information in the PI.

Step	Action
1	On the **MAC Pairs** tab, select from the list the MAC pair to edit.
2	Click **Edit**. **Result:** The *PI MACS screen* (on page 55) fields will be populated with the data for the selected MAC pair record. See *MAC Pairs fields* (on page 54) for a description of each field.
3	Change the MAC pair details as required.
4	Click **Save**. **Result:** The details are saved to the database.
5	Soft restart the PI. For details, see *Soft PI Restart* (on page 76). **Result:** The updated configuration details will be loaded by the PImanager.

Deleting MAC Pairs

Follow these steps to delete a MAC pair from the PI.

Step	Action
1	On the **MAC Pairs** tab, select from the list the MAC pair to delete.
2	Click **Delete**. **Result:** The Delete Confirmation screen displays.
3	Click **OK**. **Result:** The MAC pairs are removed from the database.
4	Soft restart the PI. For details, see *Soft PI Restart* (on page 76). **Result:** The updated configuration details will be loaded by the PImanager.

PI Users

Introduction

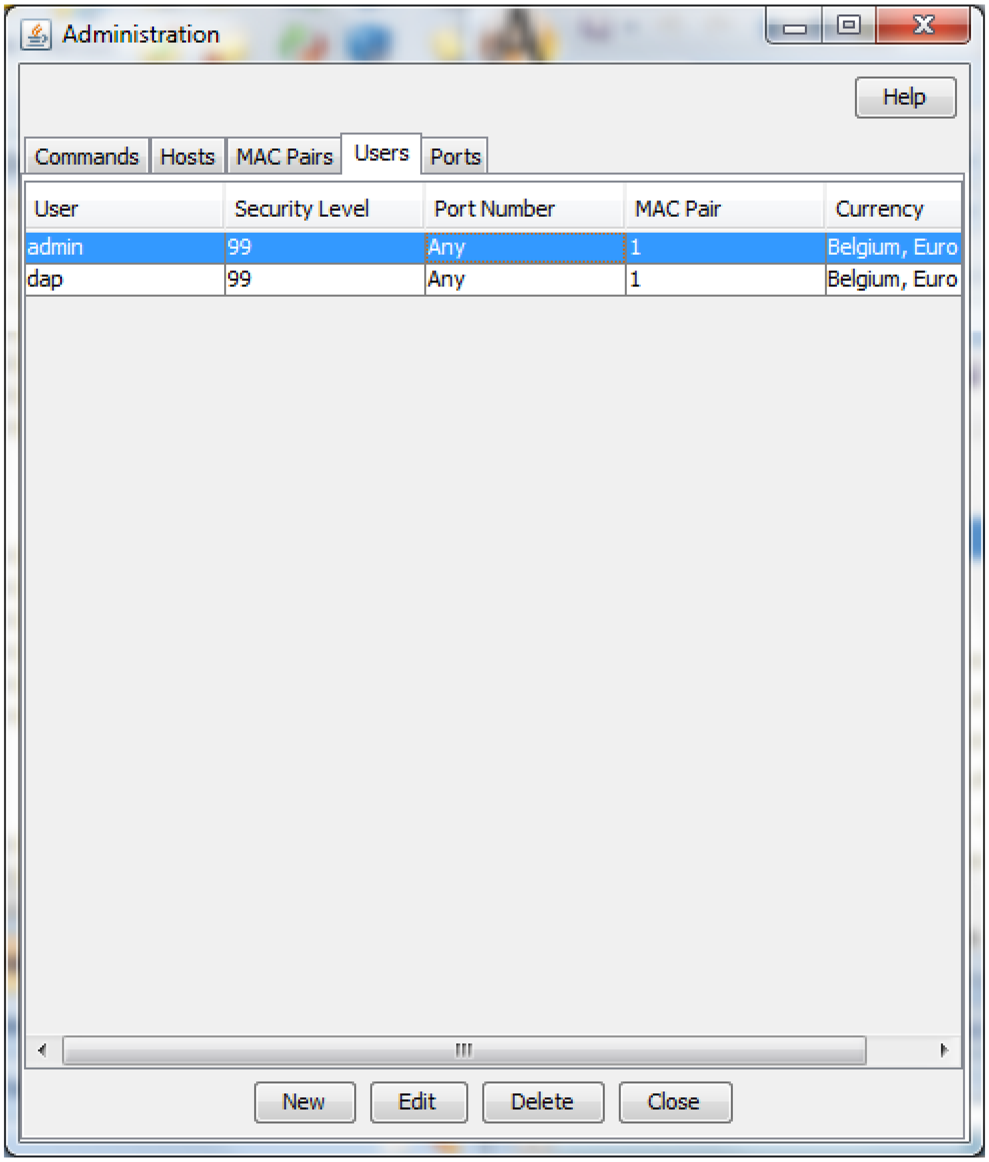
The **Users** tab of the PI Administration screen enables you to add new PI users and to edit and delete existing PI users.

When you add a new PI user you select the service providers to associate with the user. The PI user can run PI commands only for those service providers. This allows you to restrict the data that the PI user can query or modify through the PI. The PI returns a NACK if a PI user attempts to run a PI command for a service provider that they are not associated with.

In addition, you specify the connection details and security level of the PI user. The first command sent to the PI by the PI user will be a connect command, specifying the username and password. PI users can access only those commands that have a security level less than or equal to their security level. Users can use only the MAC pair specified in their profile and are restricted to using the port specified on the screen.

Users tab

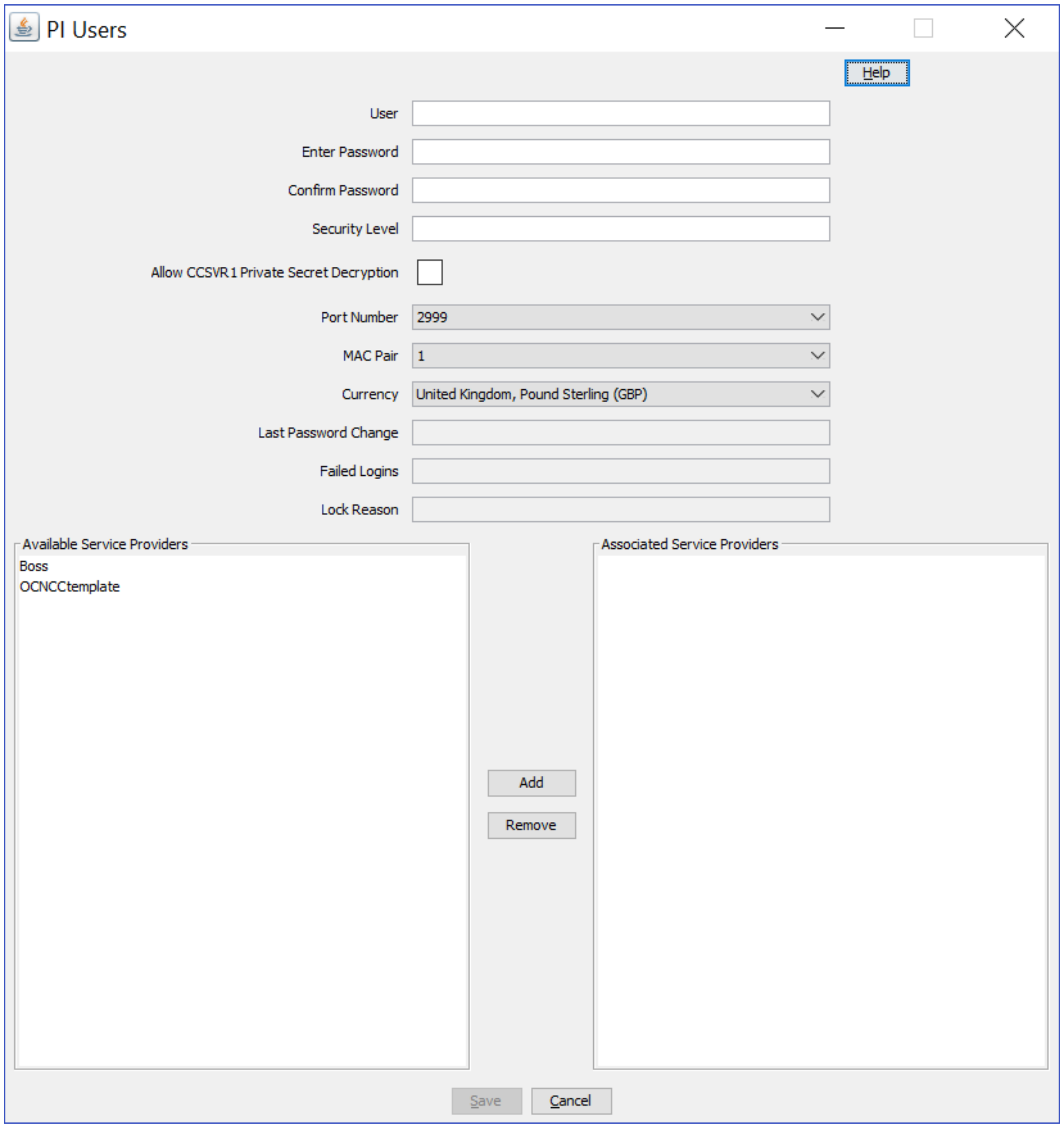
The following example screen shows the **Users** tab in the PI Administration screen.



User	Security Level	Port Number	MAC Pair	Currency
admin	99	Any	1	Belgium, Euro
dap	99	Any	1	Belgium, Euro

PI Users screen

The following example screen shows the PI Users screen.



Users fields

The following table describes the function of each field in the PI Users screen.

Field	Description
User	The unique username for this user. **Note:** This field cannot be changed after it is first saved.
Enter Password	Sets the password for this PI user.
Confirm Password	Confirms the user's password.

Field	Description
Security Level	The security level for this user. Specify a value between 1 and 99 (inclusive). The user will be able to run PI commands with security levels equal to or lower than this number.
Allow CCSVR1 Private Secret Decryption	Permission for the user to decrypt voucher private secret to obtain HRN.
Port Number	The port number this user can connect from.
MAC Pair	The MAC pair this user can connect from. MAC pairs are the security keys to encode and decode encrypted data.
Currency	The reporting currency for this user.
Last Password Change	Date of the last successful password change for this PI User.
Failed Logins	The number of login failures since the last successful login.
Lock Reason	Displays the reason that a user has been locked out of the system. You can unlock a locked PI user by clearing this field. For more information, see Unlocking PI users from UI. If a user fails to log in to the system in three successive attempts, the system locks the account and the following text is displayed: `LOCKED: Failed login, maximum attempts exceeded.` For more information about locked accounts, see Unlocking locked PI users. **Warning:** When you create a user, leave this field blank to avoid creating a locked account.
Available Service Providers	The list of service providers that you can associate with this user.
Associated Service Providers	The list of service providers associated with this user. For PI commands that allow a service provider to be specified, the data that this user can update or query through the PI is restricted to data that is managed by a service provider in this list.

Adding PI users

Follow these steps to add a new PI user.

Step	Action
1	On the **Users** tab, click **New**. **Result:** The PI Users screen appears. See *Users fields* (on page 58) for a description of each field.
2	In the **User** field, type a unique username for the PI user you want to add.
3	In the **Enter Password** field, type the user's password.
4	In the **Confirm Password** field, retype the user's password to confirm.
5	In the **Security Level** field, type the command security level for this user. Specify a value between 1 and 99 (inclusive). The user will be able to run PI commands with security levels equal to or lower than this number.
6	From the **Port Number** list, select the port the user can connect from. To allow the user to connect from any port, select `Any`.
7	From the **MAC Pair** list, select the MAC pair the user will connect from.

Step	Action
8	From the **Currency** list, select the reporting currency for the user.
9	Add the service providers the PI user will be able to run PI commands for to the list of associated service providers: • To add a service provider to the list, select the service provider in the **Available Service Providers** box and click **Add**. • To remove a service provider from the list, select the service provider in the **Associated Service Providers** box and click **Remove**.
10	Click **Save**. **Result:** The new user details are saved in the database.

Editing PI users

Follow these steps to edit the details of a PI user.

Step	Action
1	From the list of PI users on the **Users** tab, select the user whose details you want to edit.
2	Click **Edit**. **Result:** The PI Users screen is populated with the data from the selected user record. See *Users fields* (on page 58) for a description of each field.
3	Change the user details as required.
4	Click **Save**. **Result:** The details are saved to the database.

Deleting PI users

Follow these steps to delete a PI user.

Step	Action
1	From the list of PI users on the **Users** tab, select the user you want to delete.
2	Click **Delete**. **Result:** The Delete Confirmation dialog box appears.
3	Click **OK**. **Result:** The PI user is removed from the database.
4	Soft restart the PI. For details, see *Soft PI Restart* (on page 76). **Result:** The updated configuration details will be loaded by the PImanager.

Unlocking PI users from UI

Follow these steps to unlock a locked PI user.

Step	Action
1	From the list of PI users on the **Users** tab, select the locked PI user.
2	Click **Edit**. **Result:** The PI Users screen is populated with the data from the selected user record. See *Users fields* (on page 58) for a description of each field.
3	Clear the **Lock Reason** field.
4	Click **Save**.

Step	Action
	Result: The PI user is unlocked.
5	Soft restart the PI. For details, see *Soft PI Restart* (on page 76).
	Result: The updated configuration details will be loaded by the PImanager.

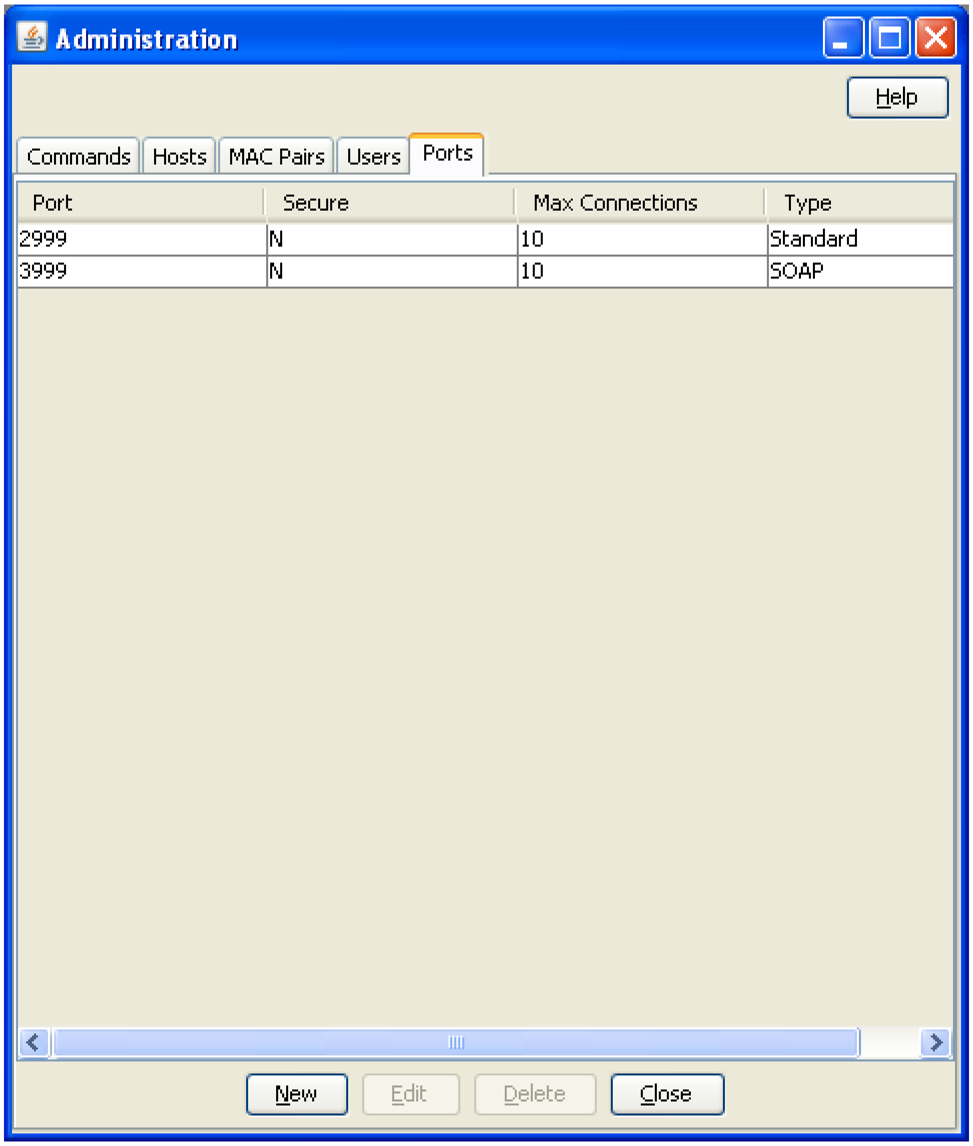
PI Ports

Introduction

The **Ports** tab of the PI Administration screen enables the configuration of the ports the PIprocesses listens on.

Ports tab

Here is an example **Ports** tab.



Port	Secure	Max Connections	Type
2999	N	10	Standard
3999	N	10	SOAP

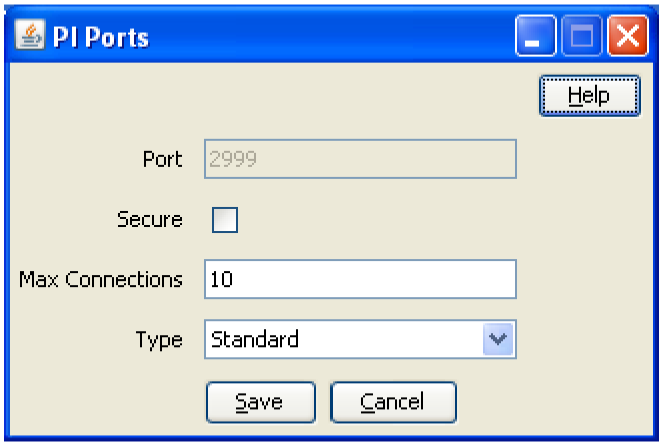
Ports fields

This table describes the function of each field.

Field	Description
Port	The unique port number which will have a PIprocess listening on it. **Note:** This field cannot be changed after it is first saved.
Secure	If **Y**, the port will be secure. If **N**, the port will be insecure.
Max. Connections	The maximum number of concurrent connections to the port.
Type	The type of PI commands which can be run on this port.

PI Ports screen

Here is an example PI Ports screen.



Port: 2999
Secure: ☐
Max Connections: 10
Type: Standard

Adding ports

Follow these steps to add new ports to the PI.

Step	Action
1	On the **Ports** tab, click **New**. **Result:** The PI Ports screen appears. See *Ports fields* (on page 62) for a description of each field.
2	Enter in the **Port** field the port number.
3	Select the **Secure** check box if this port should be secure. Deselect the **Secure** check box if this port is not required to be secure.
4	In the **Max. Connections** field, type the maximum number of concurrent connections this port will support.
5	From the **Type** list, select the type of commands that can be run on this port.
6	Click **Save**. **Result:** The new port details are saved in the database.
7	Hard restart the PI. See *Hard PI Restart* (on page 76). **Result:** The new configuration details are loaded by the PImanager.

Editing ports

Follow these steps to edit port information in the PI.

Step	Action
1	On the **Ports** tab, select the port you want to edit.
2	Click **Edit**. **Result:** The PI Ports screen is populated with the data from the selected port record. See *Ports fields* (on page 62) for a description of each field.
3	Change the port details as required.
4	Click **Save**. **Result:** The details are saved to the database.
5	Hard restart the PI. See *Hard PI Restart* (on page 76). **Result:** The new configuration details are loaded by the PImanager.

Deleting ports

Follow these steps to delete a port from the PI.

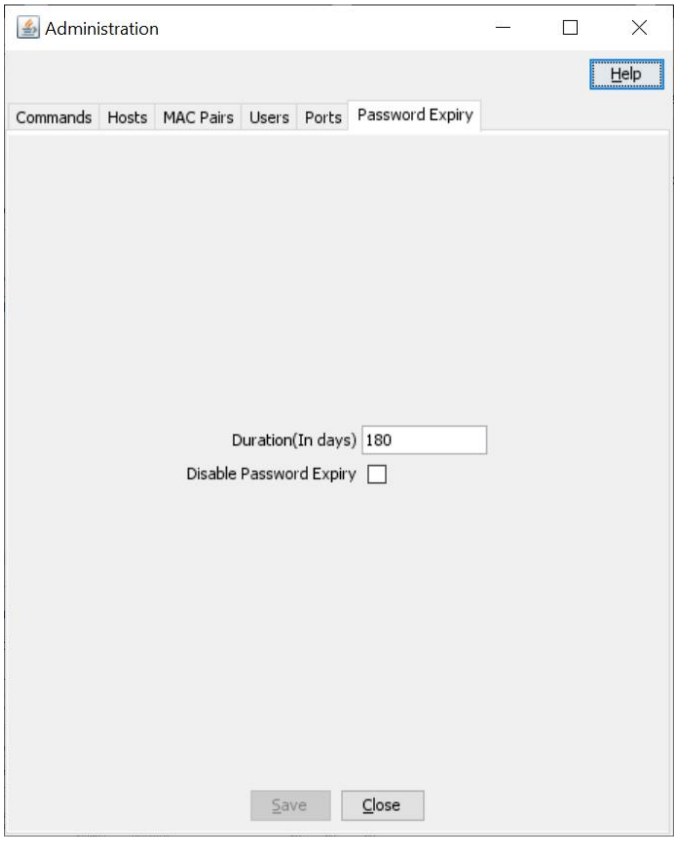
Step	Action
1	On the **Ports** tab, select the port to delete.
2	Click **Delete**. **Result:** The Delete Confirmation dialog box appears.
3	Click **OK**. **Result:** The port is removed from the database.
4	Hard restart the PI. See *Hard PI Restart* (on page 76). **Result:** The new configuration details are loaded by the PImanager.

PI Password Expiry

Introduction

The **Password Expiry** tab of the PI Administration screen enables you to configure the number of days after which PI user's password should expire. You can also disable password expiration from this tab.

Password Expiry Tab



Duration(In days) 180
Disable Password Expiry ☐

Password Expiry Fields

This table describes the function of each field.

Field	**Description**
Duration(In days)	Number of days after which password should expire.
Disable Password Expiry	Selecting this checkbox disables password expiry and password will never expire for PI users.

Configuring Password Expiry

Follow these steps to configure password expiry for PI users.

Step	**Action**
1	Click on the **Password Expiry** tab.
2	In the **Duration(In days)** field, enter the number of days after which the password should expire. **Note**: Select **Disable Password Expiry** checkbox if you do not want the PI user password to expire.
3	Click **Save**. **Result:** The password expiry configuration is saved.
4	Hard restart the PI. See *Hard PI Restart* (on page 76).

Step	Action
	Result: The new configuration details are loaded by the PImanager.

Chapter 4

PI Tester Screen

Overview

Introduction

This chapter explains how to use the PI Tester for standard ports screen.

In this chapter

This chapter contains the following topics.

PI Tester Screen

Introduction

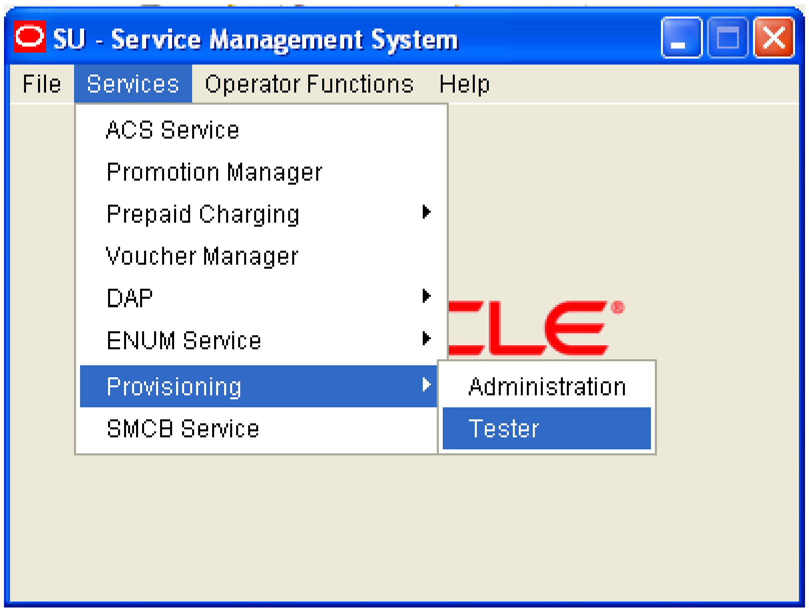
Use the PI Tester for standard ports screen to check that the PI commands are returning the correct results. It contains the following tabs:

- General
- Management
- Connection

Accessing the PI Tester screen

Follow these steps to access the PI Tester for standard ports screen.

Step	Action
1	Select the **Services** menu from the Service Management System main menu.

Step	Action
	
2	Select **Provisioning**.
3	Select **Tester**.
	Result: You see the PI Tester for standard ports screen.

General

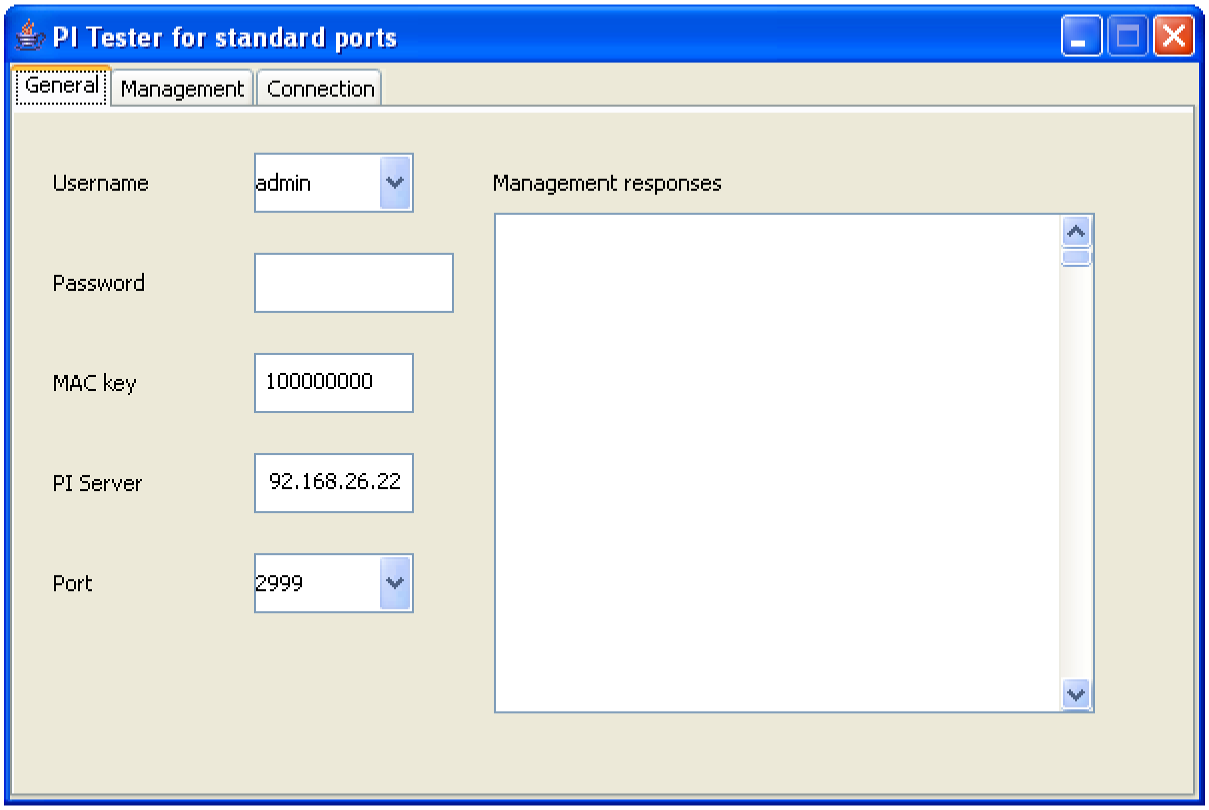
Introduction

Use the **General** tab of the PI Tester for standard ports screen to modify general test attributes such as the test user and MAC address, and to view the results of management commands.

Note: The fields on the **General** tab are populated automatically. You only need to change them if you do not want to use the default value for a field.

General tab

Here is an example **General** tab.



Username: admin
Password:
MAC key: 100000000
PI Server: 92.168.26.22
Port: 2999

General fields

This table describes the function of each field.

Note: These fields are automatically populated with the default values.

Field	**Description**
Username	The user carrying out the tests. The user, defined in the Users tab of the Administration screen, must have the appropriate permissions. Typically, the admin user is used.
Password	The password for the user.
MAC key	The MAC key to use. This ensures the connection is secure.
PI Server	The PI server to connect to.
Port	The port to connect to. The list of available ports only includes port numbers defined for the PI server the screens came from.
Management responses	Displays the results of management commands entered on the **Management** tab of the PI Tester screen.

Editing the General tab

Follow these steps to change the general test attributes.

Step	Action
1	Select the **General** tab on the PI Tester for standard ports screen.
2	Change the values for the general attributes as required.

Management Tests

Introduction

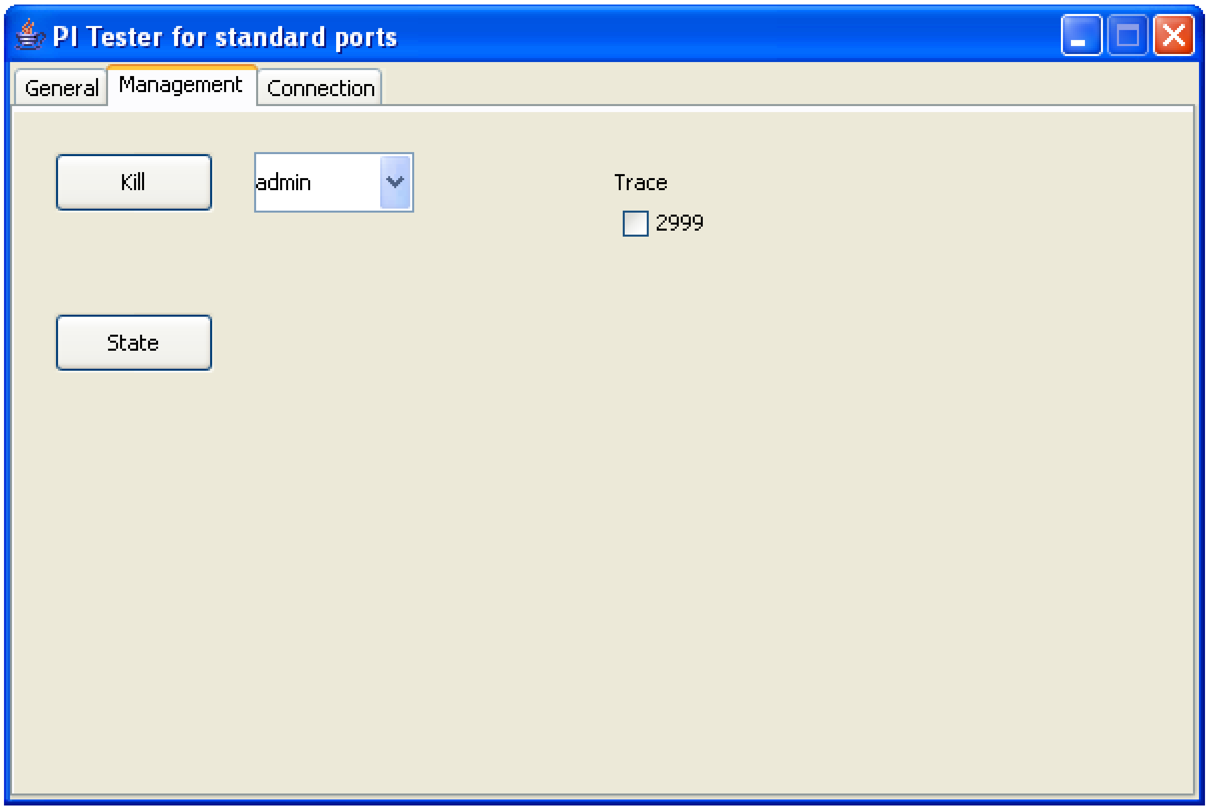
Use the **Management** tab in the PI Tester for standard ports screen to send management commands to the PI server, using the values from the **General** tab. The following commands are available:

- Kill – To kill the PI connection for a selected user
- State – To see the current state of the PI commands, hosts and users
- Trace – To set up a trace on the port specified in the **General** tab

Note: The test responses are reported on the **General** tab.

Management tab

Here is an example **Management** tab.



Management fields

This table describes the function of each field.

Field	Description
User selection box	Lets you select a user from the drop down list. You can then kill the user's connection by clicking **Kill**.
Trace	Lets you switch tracing on, for the port specified on the **General** tab. The trace log, PI<port>.log is saved to the following directory: • if the PImanager was started with the inittab or the startup script, it is saved in `/IN/service_packages/SMS`. • if the PImanager was started manually, it is saved in `/IN/service_packages/MOB_PP/bin`.

Using management tests

The following steps explain how to use the management tests.

Step	Action
1	To kill a user's connection, select the user from the list, and click **Kill**.
2	To find out the state of commands, hosts and users, click **State**.
3	To put a trace on the port currently selected in the **General** tab, select the **Trace** box.

Connection tests

Introduction

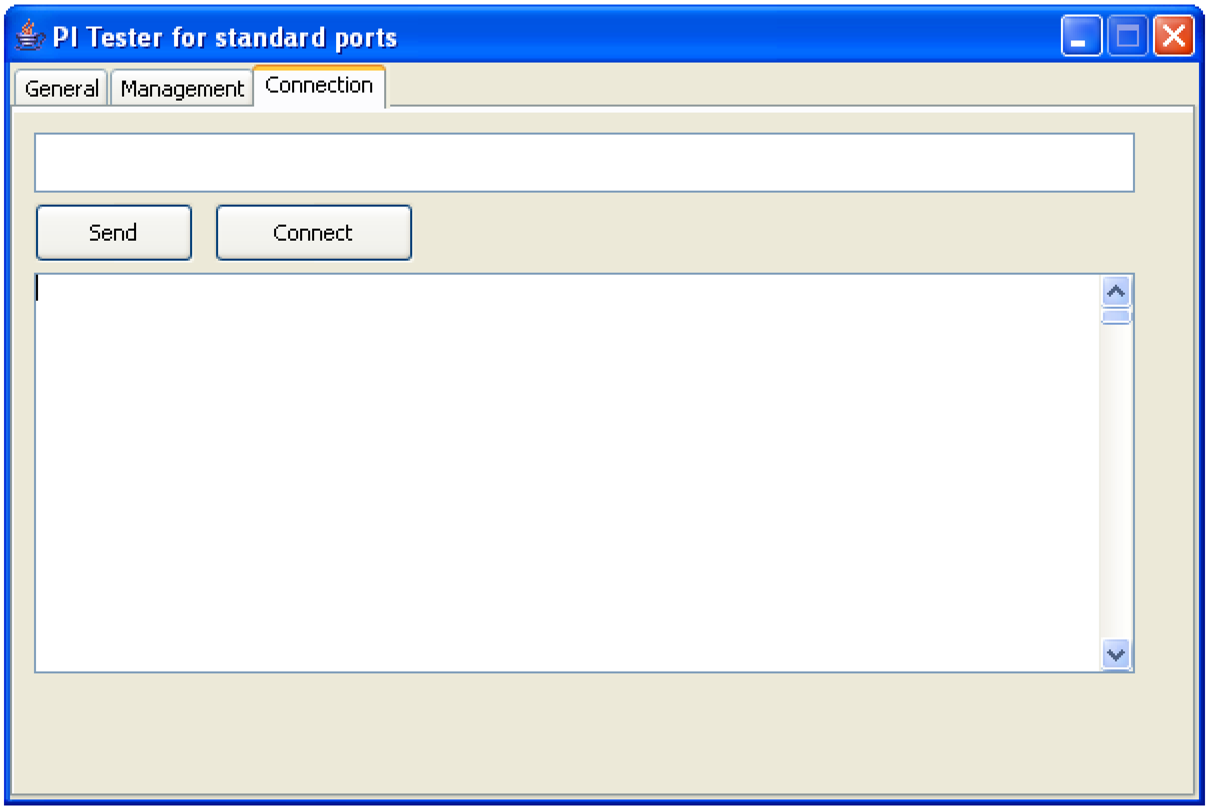
Use the **Connection** tab on the PI Tester for standard ports screen to enter commands directly, and check the results. The results appear in the results window on the **Connection** tab.

For PL/SQL commands, the parameters must be entered in the correct order. For C commands, the required parameters must be entered before the optional parameters. See your specific commands specification for details on the syntax to use for commands and the expected results.

Note: The commands you specify may alter the database, so you should use this facility with caution, especially when testing on a live database.

Connection tab

Here is an example **Connection** tab.



Using connection tests

Follow these steps to run tests from the **Connection** tab on the PI Tester for standard ports screen.

Step	Action
1	Click **Connect** to set up a connection to the PI server specified on the **General** tab in the PI Tester for standard ports screen. **Result:** The results of the connection attempt, including the systamp that will be used later, appear in the lower window.
2	In the upper text box, type in the commands you want to test, and click **Send**. You must use the following format: `COMMAND=ACTION:REQUIRED_PARAMETER=VALUE,REQUIRED_PARAMETER=VALUE,OPTIONAL_PARAMETER=VALUE,OPTIONAL_PARAMETER=VALUE,SYNSTAMP=NUMBER` **Result:** The commands are sent to the PI process and the results appear in the lower window.
3	Click **Disconnect** to disconnect from the PI server.

Note: There is a continuous password age check when PI user is connected for a long period. During a persistent connection:

a. If the password for PI user reaches grace period, PIprocess writes warning message in PImanager.log everyday about the number of days remaining for the password to expire until it reaches the end of the grace period.

b. If password expires, PIprocess throws an error message and the session gets terminated. For example, "CRITICAL:[Password Expired]". You need to change the password or clear the lock reason for that PI user from the PI User screen, to reconnect.

Chapter 5

Background Processes

Overview

Introduction

This chapter explains the PI processes which are used. The PImanager is started using the inittab, and the PIbatch process is started manually.

In this chapter

This chapter contains the following topics.

PImanager

Purpose

PImanager starts and stops PI processes as required.

Startup

PImanager can be started by either of two methods:

inittab script

The PImanager should normally be started from the **inittab** script.

/IN/service_packages/PI/bin/PImanagerStartup.sh

If PImanager is started this way, the output will be shown on the screen, rather than put in the log file.

Started directly

The process can be started directly, using the following code:

```
$ su - smf_oper
$ cd /IN/service_packages/PI/bin
$ ./PImanager [-u <user/password>] [-S <Y|N>] [-t <n>] [-M <m>] [-h] &
```

If PImanager is started this way, the output will be shown on the screen, rather than put in the log file.

Optional parameters

This table describes the optional parameters.

Parameter	Default	Description
-u *username/password*	/	The Oracle username and password.
-S *Y|N*	Y	Turn on synstamp processing.
-t *n*	no timeout	Set the timeout to *n* seconds.
-M *m*	use database setting	Override database settings for the *PIprocess* (on page 77) mode of operation • 0=Standard • 1=XML with command mode • 2=XML session mode • 3=SOAP
-h		Display usage and exit.

Shutdown

To stop the PImanager when it is running from the **inittab** file, use the utility script (as root user):

```
/IN/service_packages/PI/bin/PIstop.sh.
```

This will also stop all PIprocesses.

To restart after stopping with PIstop.sh, use the utility script (as root user):

```
/IN/service_packages/PI/bin/PIstart.sh.
```

To stop the PImanager when not running from the **inittab** file, send the PImanager a **kill -TERM** signal. This will stop the PImanager and any associated PIprocesses.

Reinitializing the PImanager

The PI can be re-initialized using either a “hard” or “soft” reset. The preferred way should be to use the soft reset.

Soft PI Restart

A soft reset is performed by using the utility script:

```
/IN/service_packages/PI/bin/PIreread.sh
```

This causes the PImanager to instruct the PIprocesses to re-read the database. However, the PIprocesses will not re-read the database until all connections have been dropped.

Note: This will not cause the PImanager to start any new PIprocesses added using the *PI Ports* (on page 61) screen. A hard reset must be done in this case.

Hard PI Restart

To do a hard reset, for example, if new PIprocesses have been added using the PI Ports screen, use the script:

```
/IN/service_packages/PI/bin/PIrestart.sh
```

This terminates the PImanager and all PIprocesses, and the inittab will then restart them.

Note: All connections to the PIprocesses will be lost.

Failure

If the PImanager fails, no commands will be processed. All PIprocesses will also fail.

Output

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

PIprocess

Purpose

The PIprocess waits for TCP/IP connections, and processes commands sent to it. These commands can be management commands, such as "Connect", "Status" and "Disconnect", or PIcommands, such as "Query Subscriber".

The PIprocess checks that the user and remote host are valid, and then processes the command, either loading the command from a shared library, or executing a PL/SQL function in the database.

Startup

PIprocesses are started by the PImanager process.

If PIprocesses are added using the PI Port screens, the PImanager must be hard restarted to start the new PIprocess processes.

Shutdown

PIprocesses are shut down or restarted using the command scripts for the PImanager which started them, or by sending it a **kill -TERM** signal.

Reinitialising a PIprocess

To force a PIprocess to re-read the database, send it a **kill -HUP** signal. The PIprocess will re-read the database when all connections to it have closed. It is preferable to re-initialize the PImanager (see above) rather than individual PIprocesses.

Configuration

PIprocess is configured using PImanager's configuration.

Failure

If PIprocess fails, PIcommands sent to the port that PIprocess is running on will fail.

Output

The PIprocess writes error messages to the system messages file, and also writes additional output to **/IN/service_packages/PI/tmp/PImanager.log**.

PIbeClient

Purpose

The PIbeClient interacts with billing engines. It is only available for installations which include CCS. If VWS is installed, PIbeClient will connect to the beServer on the Voucher and Wallet Server.

CCS process

PIbeClient is installed by the piCcsSms package. It will only be available on your system if you have installed CCS.

Startup

PIbeClients are started by the PImanager process as needed.

Shutdown

PIbeClient are shutdown or restarted using the command scripts for the PImanager which started them, or by sending it a **kill -TERM** signal.

Configuration

The PIbeClient is configured in the `PI` section of **eserv.config**. For more information see *PIbeClient* (on page 10).

Output

The PIbeClient writes error messages to the system messages file, and also writes additional output to **/IN/service_packages/PI/tmp/PImanager.log**.

PIbatch

Purpose

The PI batch program is a utility to allow multiple PI commands to be sent to PIprocesses, with the commands being specified in a file.

Startup

PIbatch is started with the command line:

```
PIbatch -D script server
```

Configuration

The PIbatch accepts the following command line arguments.

Usage:

```
PIbatch script server
```

PIbatch input files

The PIbatch input scripts contain one command per line. You must order the commands as follows:

1. `!c` *`port user password`* `[`*`synstamp`*`] [`*`mac`*`]` – To instruct the PIbatch to connect to the PI on the given port number, using the specified username and password, with the optional synstamp, and with the MAC provided.
2. List the PI commands and PIbatch commands. The MAC and SYNSTAMP for commands are supplied automatically.

   Note: If you place a ";" on the command line, you will have to include the MAC and SYNSTAMP in the command.
3. `!d` – To instruct the PIbatch to disconnect from the PI.

   Note: When you place the $ character at the beginning of a line, it is executed as a shell command.

Example input file

This is an example PIbatch input file.

```
!c 2999 admin admin 151111111
debug on
state
CCSCD4=CHG:MSISDN=1473111222,ADD=1234
!d
```

Note: The MAC address in the "!c" connection string is constructed from the data in the MAC Pairs tab on the PI Administration screens. It is the MAC Pair number prefixed to either the first or second MAC address, as required. In this example, the MAC Pair number is 1 and the MAC Address is 51111111.

Failure

If PIbatch fails, the commands in the batch file will not be executed. Individual commands in the batch file can also fail.

Output

The results of the PI batch program are placed in a file. The file has the same name as PIbatch input file and a **.result** file extension.

The following text appears in the output file for each command sent to the PI:

"Running command *command_name the_command_result* End of output from *command_name*"

The output file can also contain some of the following information:

- the -> symbol followed by text sent to the PI,
- the -< symbol followed by text received from the PI,
- comments from the input script,
- and when the PIbatch disconnects from the PI, the word "Disconnected" is written to the output file.

Results file example

This is an example PIbatch results file.

```
->admin,admin;
<-ACK,SYNSTAMP=2005021010342483;
->CCSCD1=DEL:MSISDN=1107, SYNSTAMP=2005021010342484,MAC=135424;
<-CCSCD1=DEL:DELETEUser:NACK:1-MSISDN 1107 is not
valid,SYNSTAMP=2005021010342484,MAC=114357;
Disconnected
```

PIbatch XML

Purpose

For PIbatch, the XML formatted PI commands are read from an input file, sent to the PIprocess, and results are optionally returned to a results file.

Startup

PIbatch_xml is started with the command line:

```
PIbatch_XML [-u username] [-p password] [-h hostname] [-n port_number] [-l loop] [-t
throttle] [-f trace_file] [file...]
```

Configuration

The PIbatch_xml accepts the following command line arguments.

Usage:

```
PIbatch_XML [-u usr] [-p pwd] [-h host] [-n port] [-l loop] [-t throttle] [-f
trace_file] [-M mode] [-c chunk] [-i implicit] [-S ssl] [file...]
```

The available parameters are:

Parameter	Default	Description
-u *usr*	admin	The username that should be used for logging into the PIprocess.
-p pwd	admin	The password that should be used when logging into the PIprocess.
-h *host*	localhost	The host name of the PI server.
-n *port*	2999	The port number of the PIprocess.
-l *loop*	1	How many times to loop through the commands.
-t *throttle*	no limit	Maximum number of requests per second.
-f *trace_file*	no file	File to append the PI responses to.
-M *mode*	1	Mode of operation • 1=XML • 2=SOAP
-c *chunk*	Y	Stipulate chunking (Y|N) in the incoming documents (document is in sections preceded by length parameters, and terminated by a single zero on the last line).
-i *implicit*	Y	Request implicit login (if Y, do not send initial Login request since the document is assumed to contain user credentials).
-S *ssl*	Y	Request SSL (secure) operation (value Y creates secure connection)
file	standard input	The input file. More than one input file can be specified.

PIbatch xml input file

The input file format is a list of XML formatted PI commands. Each input file can have more than one command. There may be more than one input file.

```
<?xml version="1.0"?>
<methodCall>
    <methodName>PI.OP</methodName>
    <params>
        <param><value><string>@TOKEN@</string></value></param>
        <param><value><string>command_name</string></value></param>
        <param><value><string>action_name</string></value></param>
        <param><value><struct>
            <member>
                <name>param_name</name>
                <value><string>param_value</string></value>
            </member>
        </struct></value></param>
    </params>
</methodCall>
```

@TOKEN@: Will be replaced with the authentication token by the PIbatch_XML program.

PIbatch_XML supports sending/receiving SOAP requests/responses. Here is an example SOAP request.

PIbatch_XML SOAP input file example:

```
    <env:Envelope
        xmlns:env="http://schemas.xmlsoap.org/soap/envelope/"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xmlns:enc="http://schemas.xmlsoap.org/soap/encoding/"
        xmlns:ns1="http://uk.oracle.com/pi">
        <env:Body>
            <ns1:command_name>
                <ns1:AUTH>@TOKEN@</ns1:AUTH>
                <ns1:param_name1>param_value</ns1:param_name1>
                <ns1:param_name2>param_value</ns1:param_name2>
                <ns1:param_name3>param_value</ns1:param_name3>
                etc...
            </ns1:command_name>
        </env:Body>
    </env:Envelope>
```

Where *command_name* and *param_name* would be substituted with the actual PI command and parameters to be executed.

SOAP Example:

```
    <env:Envelope
        xmlns:env="http://schemas.xmlsoap.org/soap/envelope/"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xmlns:enc="http://schemas.xmlsoap.org/soap/encoding/"
        xmlns:ns1="http://uk.oracle.com/pi">
            <ns1:CCSCD1_ADD>
                <ns1:AUTH>@TOKEN@</ns1:AUTH>
                <ns1:MSISDN>6122000193</ns1:MSISDN>
                <ns1:PROVIDER>Boss</ns1:PROVIDER>
                <ns1:PRODUCT>nzcl</ns1:PRODUCT>
                <ns1:CHARGING_DOMAIN>5</ns1:CHARGING_DOMAIN>
            </ns1:CCSCD1_ADD>
        </env:Body>
    </env:Envelope>
```

Output

The output file is simply written with the responses from the PI.   See the relevant PI command definition for details.

PIuser

Purpose

Use the PIuser utility to create new PI users and passwords to enable users to log into the PI remotely.

For security reasons, before a PI user can run PI commands for a service provider you must associate the service provider with the PI user. This will ensure that the PI user is able to modify and query the data only for those service providers that they are associated with.

To associate a service provider with a PI user, edit the PI user details on the **PI Users** tab in the PI Administration screen. For more information, see *PI Users* (on page 56).

Startup

Start PIuser from the command line by using the following sytax:

```
PIuser -s security_level [-d db_login] [-u username] [-p password] [-n port] [-m
mac_pair] [-c currency_code]
```

Configuration

The following table describes the PIuser command line parameters.

Parameter	Description
-s *security_level*	The PI security level for the new PI user. The new PI user will be able to run PI commands with security levels equal to or lower than this number. Specify a number between 1 and 99 inclusive.
-d *db_login*	(Optional) The username and password for the Oracle database login ID. Defaults to / if not set.
-u *username*	(Optional) The username for the new PI user. PIuser prompts for the username if not set. You must specify a unique name.
-p *password*	(Optional) The password for the new PI user. PIuser prompts for the password if not set and checks for strength based on the password policy that is set. For admin users, this password can be set at installation time from OUI invoking the PIuser command and skips strength checking of the password.
-n *port*	(Optional) The port number that the PI user will use for remote login. Defaults to all ports if not set.
-m *mac_pair*	(Optional) The MAC pair the new PI user can connect from. Defaults to 1 (one) if not set.
-c *currency_code*	(Optional) The reporting currency for the new PI user. Defaults to the system currency if not set.

Chapter 6

PI Management Commands

Overview

Introduction

This chapter explains the Oracle Communications Convergent Charging Controller Provisioning Interface (PI) management commands.

In this chapter

This chapter contains the following topics.

Debug Command

Purpose

The PI can run in debug mode. You should use this mode only to trace faults.

Debug mode can be turned on or off for each component or command while the PI is running. To do this, in the PItester screen, send a debug command to the PIprocess.

The general list of components that can be specified is:

- PIbatch (turn debug on in PIbatch)
- PImanager (turn debug on in PImanager)
- PIprocess (turn debug on in non-command parts of PIprocess)
- PIcCommands (turn debug on in common parts of PI commands)

If piCcsSms is installed:

- PIbeClient (turn debug on in non-beClientIF parts of PIbeClient)

For the 3 different types of PI command syntax only the standard and XML currently support the ability to dynamically turn debug on and off for specified PI commands. PI XML SOAP does not currently support this functionality and requires a restart of the PI to turn debug on or off.

Note: Refer to your specific *Command Specification* for details of all the available commands.

Format

The format of the debug command is:

```
debug {on/off} component1 [component2] [component3] ... [component];
```

Component is the command, or command with the equals sign converted to an underscore.

Example:

- CCSCD1_ADD
- CCSCD1_DEL

The format of the dynamic debug for XML commands is:

Example PI XML debug commands:

```
<?xml version="1.0"?>
<methodCall>
   <methodName>PI.Debug</methodName>
      <params>
         <param>
            <value>
               <string>@TOKEN@</string>
            </value>
         </param>
         <param>
            <value>
               <struct>
                  <member>
                     <name>component1</name>
                     <value>
                        <string>off</string>
                     </value>
                  </member>
[
                  <member>
                     <name>component2</name>
                     <value>
                        <string>on</string>
                     </value>
                  </member>
...
]

               </struct>
            </value>
         </param>
      </params>
</methodCall>
```

Example

The following command examples turns debug on for CCSCD1=ADD:

```
debug on CCSCD1_ADD;
```

To turn on debug for CCSCD1=ADD and CCSCD1=DEL commands, use:

```
debug on CCSCD1_ADD CCSCD1_DEL;
```

To turn on debug for all CCSCD1 commands, use

```
debug on CCSCD1
```

Note: This also turns on PIcCommands but only for the command used, that is, CCSCD1_ADD.

The following PI XML command example turns debug on for CCSCD1=ADD and off for CCSCD1=DEL:

```
<?xml version="1.0"?>
   <methodCall>
   <methodName>PI.Debug</methodName>
      <params>
         <param>
            <value><string>@TOKEN@</string></value>
         </param>
         <param>
            <value>
               <struct>
                  <member>
                     <name>CCSCD1=ADD</name><value><string>off</string></value>
                  </member>
                  <member>
                     <name>CCSCD1=DEL</name><value><string>off</string></value>
                  </member>
               </struct>
            </value>
         </param>
    </params>
</methodCall>
```

Note: This turns DEBUG+PIcCommands on or off, but only for the command(s) specified (CCSCD1=ADD and CCSCD1=DEL).

Output

Debug prints output to the **/IN/service_packages/PI/tmp/PImanager.log** file.

The **/IN/service_packages/PI/tmp/PImanager.log** file will only log successfully completed debug {on/off} commands for the PIprocess.

Example:

```
Oct 5 23:22:17 PIprocess:2998(21833) About to turn debug on for CCSCD1=ADD
Oct 5 23:22:17 PIprocess:2998(21833) About to turn debug off for CCSCD1=DEL
```

When dynamic PI command debug is on, the output is printed to the **/IN/service_packages/PI/tmp/PImanager.log** file.

Traceon Command

Purpose

The traceon command enables tracing of all PI commands. The tracing results are output to a file.

Tip: This command is now deprecated. We recommend you use the debug command instead.

Format

The format of the traceon command is:

```
traceon;
```

Example

The following command enables tracing of PI commands:

```
traceon;
```

Output

PI command tracing is output to the following file:

```
PIport_number.trace
```

Traceoff Command

Purpose

This command disables tracing.

Tip: This command is deprecated. We recommend you use the debug command instead.

Format

The format of the traceoff command is:

```
traceoff;
```

Example

The following command disables tracing of PI commands:

```
traceoff;
```

State Command

Purpose

Use the state command to print the current state of the PIprocess.

Format

The format of the state command is:

```
state;
```

Example

The following command prints the current state of the PI process:

```
state;
```

Output

The current state of the PI process is output to the following file:

```
PIport_number.state
```

Kill Command

Purpose

Use the kill command to kill a connection from a given username.

Format

The format of the kill command is:

```
kill username;
```

Example

The following command kills the connection from user bob:

```
kill bob;
```

Sendrate Command

Purpose

Use the sendrate command to specify the maximum number of PI commands that an individual PIprocess will send per second for processing by the billing engine. This allows each PIprocess to place only the desired load on the billing engine.

Example: Setting the sendrate for the port being used by PIbatch to the minimum rate of one, keeps its load to a minimum. This helps preserve the billing engine capacity for "live" usage.

Tip: The default sendrate can be set for all PIprocesses in **eserv.config**. For details, see **eserv.config** *Configuration* (on page 6).

Format

The format of the sendrate command is:

```
sendrate n;
```

The available parameter is:

Parameter	Default	Description
n		The number of PI commands to send to the billing engine per second. This must be a whole number. **Tip:** To turn throttling off, set the sendrate to 0 (zero).

Checking the sendrate

You can use the state command to check the sendrate for the PI processes. This reports the values for all connected PI processes. For details see *State Command* (on page 86).

PIbatch sendrate

You can use the sendrate command to control the sendrate when using PIbatch. You:

1. Add the sendrate command to the batch input file after the connect line
2. Add a second sendrate command before the disconnect line to reset the sendrate to its original value

Warning: This will affect all connections to this PIprocess.

Batch optimization

To optimize the batch, the sendrate can be calculated as follows. The resulting number must be rounded up to the nearest whole number:

sendrate = number of commands / maximum time for batch (seconds)

Example: For 40000 commands in 4 hours (14400 seconds) the sendrate = 40000 / 14400 = 2.8. The rounded up value for the sendrate is 3.

Using sendrate for performance

You can use the sendrate command to control the load put on the billing engine by the different PI processes.

This can be achieved by setting the default sendrate in **eserv.config**, and then overriding the default for each individual PIprocess requiring a different value.

To override the default values, connect to each PIprocess port in turn and set the sendrate to the desired value.

Example

In this example there are three PIprocesses running on ports 2999, 3000, 3001. PIbatch is running on port 3001 and it needs to run at a lower rate than the other processes so that it does not overload the billing engines. A higher rate is required for the process running on port 2999. This can be achieved in the following way:

Set the throttling parameter in **eserv.config** as:

```
pi = {
    throttling = {
        sendRate = 2
    }
}
```

Then in a PIbatch script define the sendrate for ports 3001 and 2999:

```
# start of PIbatch script
# set PIbatch port to lower rate
!c 3001 PIuser PIpassword mac_number
sendrate 1
state
!d

# set shop port 2999 to higher rate
!c 2999 PIuser PIpassword mac_number
sendrate 3
state
!d
# end of PIbatch script
```

Logstats on/off Command

Purpose

Use the logstats on or off command to switch the output of the timing statistics on, or off, every 30 seconds.

The following five statistics are collected for each command, over a 30 second time period:

- Number of successful uses of the command
- Number of unsuccessful uses of the command
- Minimum response time (in milliseconds)
- Average response time (in milliseconds)
- Maximum response time (in milliseconds)

Note: These statistics are for a single period; they are not cumulative.

Output

The output from the statistics has the following format:

command name followed by the five statistics separated by a slash (/) character.

COMMAND=ACTION 12/13/1/2/3

Example Output:

```
Jul 31 15:30:59 PIprocess:2999(1001) Statistics for last 30 seconds (<command>
<successful>/<failed>/<min>/<max>/<avg>):
Jul 31 15:30:59 PIprocess:2999(1001)  ACSCLI=ADD 0/0/0/0/0 ACSCLI=DEL 0/0/0/0/0 ACSCLI=QRY
0/0/0/0/0
```

Chapter 7

About Installation and Removal

Overview

Introduction

This chapter provides information about the installed components for the Oracle Communications Convergent Charging Controller application described in this guide. It also lists the files installed by the application that you can check for, to ensure that the application installed successfully.

In this Chapter

This chapter contains the following topics.

Installation and Removal Overview

Introduction

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

- Convergent Charging Controller system requirements
- Pre-installation tasks
- Installing and removing Convergent Charging Controller packages

PI packages

An installation of Provisioning Interface includes the following packages, on the SMS:

- piSms
- piCluster (if installing on a clustered SMS)
- piAcsSms
- piCcsSms
- piSubscriberSms
- piVoucherSms
- piSrmSms
- piWalletSms
- piVpnSms
- piXmsSms
- npPISms

Packages and dependencies

The Prepaid Charging v3.0 on-line provisioning interface engine can be delivered in up to six packages:

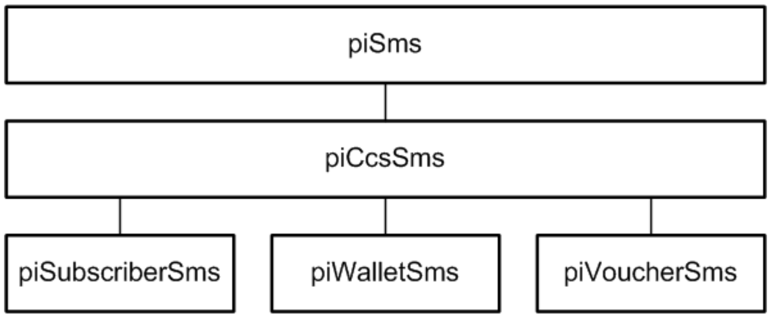
- piSms – A framework to execute a PI command.   This package is always required.

- piCcsSms – Commands and UBE client. This package is always required.
- piSubscriberSms – Command definitions for a subscriber domain. This package is optional.
- piWalletSms – Command definitions for a wallet domain. This package is optional.
- piVoucherSms – Command definitions for a voucher domain. This package is optional.
- piSrmSms – Command definitions for the subscriber relationship manager. This package is optional.

Packages must be installed as shown in the hierarchy:

1 piSms
2 piCcsSms
3 The rest in any order

Packages have the hierarchy shown in the following diagram:



Updating the eserv.config file manually

Note that at the end of the installation script of some PI packages, there is note advising the installer to manually update the **eserv.config** file using the contents of the example config file. After installation, follow these instructions to configure the **eserv.config** file. For example:

```
Example configuration files have been installed to
  /IN/service_packages/PI/etc/eserv.config.pi_example.CCS and
  /IN/service_packages/PI/etc/eserv.config.pi_example.CCS.full
Please use these as a guide to setting up your runtime
configuration file at
/IN/service_packages/eserv.config

for example. If there is no existing pi section in eserv.config, copy the entire contents of the
eserv.config.pi_example.CCS file to the end of the runtime configuration file. If there is a pi
section, copy the relevant items into the pi section.
```

See *eserv.config Configuration* (on page 6) for details.

Checking the Installation

Introduction

This topic provides a list of things which should be checked to ensure the installation was successful.

Process list

When the application is running correctly, the following processes are run by smf_oper:

- PImanager
- PIprocess (if PI has been set up on more than one port, there should be as many PIprocesses as configured ports)

Note: PIbatch may also be running.

Checking the commands

When you install a PI commands package, this inserts the new commands into new rows in the PI_COMMANDS database table. After completing the installation, check this table to ensure the new commands have been added.

Database tables

The following database tables are added to the SMF database:

- PI_COMMANDS
- PI_HOSTS
- PI_PORTS
- PI_MACS
- PI_USERS