



LinqUs Provisioning Manager 4.1

Administration Guide

All information herein is either public information or is the property of and owned solely by Gemalto NV. and/or its subsidiaries who shall have and keep the sole right to file patent applications or any other kind of intellectual property protection in connection with such information.

Nothing herein shall be construed as implying or granting to you any rights, by license, grant or otherwise, under any intellectual and/or industrial property rights of or concerning any of Gemalto's information.

This document can be used for informational, non-commercial, internal and personal use only provided that:

- The copyright notice below, the confidentiality and proprietary legend and this full warning notice appear in all copies.
- This document shall not be posted on any network computer or broadcast in any media and no modification of any part of this document shall be made.

Use for any other purpose is expressly prohibited and may result in severe civil and criminal liabilities.

The information contained in this document is provided "AS IS" without any warranty of any kind. Unless otherwise expressly agreed in writing, Gemalto makes no warranty as to the value or accuracy of information contained herein.

The document could include technical inaccuracies or typographical errors. Changes are periodically added to the information herein. Furthermore, Gemalto reserves the right to make any change or improvement in the specifications data, information, and the like described herein, at any time.

Gemalto hereby disclaims all warranties and conditions with regard to the information contained herein, including all implied warranties of merchantability, fitness for a particular purpose, title and non-infringement. In no event shall Gemalto be liable, whether in contract, tort or otherwise, for any indirect, special or consequential damages or any damages whatsoever including but not limited to damages resulting from loss of use, data, profits, revenues, or customers, arising out of or in connection with the use or performance of information contained in this document.

Gemalto does not and shall not warrant that this product will be resistant to all possible attacks and shall not incur, and disclaims, any liability in this respect. Even if each product is compliant with current security standards in force on the date of their design, security mechanisms' resistance necessarily evolves according to the state of the art in security and notably under the emergence of new attacks. Under no circumstances, shall Gemalto be held liable for any third party actions and in particular in case of any successful attack against systems or equipment incorporating Gemalto products. Gemalto disclaims any liability with respect to security for direct, indirect, incidental or consequential damages that result from any use of its products. It is further stressed that independent testing and verification by the person using the product is particularly encouraged, especially in any application in which defective, incorrect or insecure functioning could result in damage to persons or property, denial of service or loss of privacy.

© Copyright 2008-9 Gemalto N.V. All rights reserved. Gemalto and the Gemalto logo are trademarks and service marks of Gemalto N.V. and/or its subsidiaries and are registered in certain countries. All other trademarks and service marks, whether registered or not in specific countries, are the property of their respective owners.

GEMALTO, B.P. 100, 13881 GEMENOS CEDEX, FRANCE.

Tel: +33 (0)4.42.36.50.00 Fax: +33 (0)4.42.36.50.90

Printed in France.

Document Reference: DOC116488H

Product version: 4.1.7

November 10, 2009

Preface		v
	Who Should Read This Book	vi
	For More Information	vi
	Contact Our Hotline	vi
Chapter 1	Administering LinqUs Provisioning Manager	1
	Starting and Stopping LinqUs Provisioning Manager	1
	The gemconnect Script	1
	The OSAGENT and TOMCAT Scripts	2
	Resetting LinqUs Provisioning Manager 4.1	3
	Configuring User Access Rights	3
	Stopping an Incoming Connector's Input Flow	4
	Stopping Provisioning of a Product	4
	Provisioning Recommendations	5
	Deployment Considerations	5
	Managing Databases	6
	Batch File Processing	6
	Standard Batch File Processing	7
	Direct Batch File Processing	11
	Maintaining Batch Files	13
	Monitoring Batch File Commands	13
	Modifying the Batchload Input Connector's Input Flow Rate	13
	Modifying the Batchload Timeframe	14
	Managing Locks	14
	The Lock File	14
	When Locks Are Stored	15
	Command Lock Manager Configuration	15
Chapter 2	Monitoring	17
Chapter 3	Log File, Audit Trail and Billing Ticket Formats	19
	Consulting the Log File, Audit Trail and Billing Tickets	19
	Logging	19
	Audit Trail	19
	Format of the Log File	19
	Format of the Audit Trail File	19
	Format of the Billing Ticket Files	20
	Command Execution CDR Record Format	20
	Batch File Connector CDR Record Format	20
	Direct File Connector CDR Record Format	21
Chapter 4	SNMP Traps and Counters	23
	The MIB File	23
	SNMP Traps	23
	Provisioning Manager Traps	25
	SNMP Counters	26

	Platform Status	26
	Provisioning Manager Counters	27
Chapter 5	Product Configuration	33
	Product Parameters	33
	Changing Product Parameter Values	33
	Provisioning Manager Product Parameters	34
	Configuration Parameters	35
	config.properties	35
	tuning.properties	36
	utility_connectors.properties and utility_connectors.xml	43
	input_connectors.properties, input_connectors_states.ini, input_connectors.xml	46
	commands.xml and commands.properties	50
Chapter 6	Error Messages and Exceptions	51
	General Error Messages	51
Appendix A	Configuring the Web Services Component	53
Index		55

Gemalto LinqUs LinqUs Provisioning Manager 4.1 is part of LinqUs Over-The-Air Suite V4.0, a fully-integrated software solution providing you with a robust, reliable and scalable infrastructure with which to manage your (U)SIM installed base.

LinqUs Over-The-Air Suite allows you to:

- Deliver new applications directly to your subscribers
- Easily adapt your service portal for diverse subscriber types
- Control your subscribers' mobile environment.

LinqUs Provisioning Manager 4.1:

- Allows you to provision the following LinqUs Over-The-Air Suite products:
 - OTA Manager 4.0
 - Service Manager 4.0
 - Subscriber Manager 4.0
 - Device Manager 4.0
 - Phonebook Backup 4.0 SP1

Note: LinqUs Provisioning Manager commands are only compatible with a Phonebook Backup product being used in standard provisioning mode. For example, LinqUs Provisioning Manager commands may fail to correctly update products that have been customized to use database keys other than the MSISDN, ICCID, or IMEI.

- Enables provisioning to be performed by:
 - SOAP API
 - Core API
 - Batch files
 - A graphical user interface
 - Automatic provisioning
- Supports the following standard provisioning commands on all LinqUs Over-The-Air Suite products:
 - **Create Card**
 - **Change SIM**
 - **Change MSISDN**
 - **Update Subscription**
 - **Delete Card**
 - **Get Information**
- In addition, LinqUs Provisioning Manager supports a number of Tool for Batchloading (TBL) provisioning commands.
- Interfaces the network operator's SI with LinqUs Over-The-Air Suite products or other external products. LinqUs Provisioning Manager 4.1 can be customized by:
 - Adding new input connectors
 - Adding new service connectors
 - Adding new provisioning commands
 - Upgrade the GUI to reflect the above mentioned enhancements
- Schedules incoming provisioning commands in the order in which they are submitted.

- Allows you to perform the following actions through the GUI:
 - Manage subscriptions
 - Manage batch files (upload, launch, replay by error code, and so on)
 - Manage the product life cycle and configuration
 - Monitor and replay commands
 - Monitor the product and its interfaces
- Assures consistency of data within the associated LinqUs Over-The-Air Suite products (by using information from the Device Manager product).
- Guarantees integrity of data following a product restart.

Who Should Read This Book

This guide is intended to be read by network or support personnel responsible for ensuring LinqUs Provisioning Manager stays running and connected to the systems being provisioned. Readers should be familiar with system administration and network concepts, including SNMP.

This guide provides information concerning:

- Basic administration tasks (starting, stopping, restarting the product, and so on).
- Advanced administration tasks (for example, tasks to perform following a system restart).
- SNMP facility and service traps and counters.
- Log file and audit trail formats.
- Monitoring the progress and history of provisioning commands.
- Product parameters.
- Error messages.

For More Information

For more information on LinqUs Provisioning Manager 4.1, see the following documents:

Document	Description
<i>Installation Guide</i>	Describes the prerequisites and procedures for installing LinqUs Provisioning Manager 4.1.
<i>User's Guide</i>	Shows how to perform tasks using the Customer Care Interface (CCI).
<i>Customization Guide</i>	Describes how to customize the product by developing input and service connectors, or creating custom provisioning commands.
<i>Command Parameters Reference Guide</i>	Details provisioning command parameters and how to invoke the commands by batch file or using the SOAP API.

Contact Our Hotline

If you do not find the information you need in this manual, or if you find errors, contact the Gemalto hotline at <http://support.gemalto.com/>.

Please note the document reference number, your job function, and the name of your company. (You will find the document reference number at the bottom of the legal notice on the inside front cover.)

Administering LinqUs Provisioning Manager

This chapter describes how to perform basic administration tasks, including how to start, stop, restart, reset, and view the status of LinqUs Provisioning Manager 4.1.

Starting and Stopping LinqUs Provisioning Manager

LinqUs Provisioning Manager is a Framework product. It can be started or stopped in the following ways:

- 1 By starting or stopped the complete platform (with the `GEMCONNECT_STARTCOREFRWK_OPTION` set to “startptf” in the `gemconnect.cfg` script).
- 2 Using commands available in the graphical user interface (GUI).
- 3 Using custom scripts (setting the `GEMCONNECT_STARTCOREFRWK_OPTION` set to “deployfwk” in the `gemconnect.cfg` script).

The `gemconnect` Script

You use the **gemconnect** script file to start, stop, restart, and view the status of LinqUs Provisioning Manager 4.1.

Note: LinqUs Provisioning Manager may share the same platform with other LinqUs Over-The-Air Suite products; in this case the script file starts and stops the entire platform.

The **gemconnect** script is installed in the `%PLATFORM_HOME%/FRWK/bin` directory. It can be run from anywhere.

Run the script as follows:

```
gemconnect [-options] {start|stop|restart|state|help}
```

Where *options* are:

-h

Display help information.

-d

Starts the Framework (billing, audit trail, logging, and so on) in a single Java Virtual Machine (JVM). If this option is not used, the components are started in separate JVMs, which can be useful during the test and development phases, but uses more memory.

-o

Attempt to start or stop the associated OSAGENT. (The OSAGENT is a process that allows CORBA servers to register their objects and assists client applications in the location of objects.)

-q

Quiet mode; do not generate log file output.

-t

Start or stop the associated Tomcat web server.

-v

Enables verbose output.

Warning: Using this mode can dramatically decrease platform performance.

Table 1 - Script Commands

Task	Command	Comment
Start the platform	<code>%PLATFORM_HOME%/FRWK/bin/gemconnect start</code>	
Stop the platform	<code>%PLATFORM_HOME%/FRWK/bin/gemconnect stop</code>	
Restart the platform	<code>%PLATFORM_HOME%/FRWK/bin/gemconnect restart</code>	
View the state of the platform	<code>%PLATFORM_HOME%/FRWK/bin/gemconnect state</code>	Displays the current status of all LinqUs Over-The-Air Suite products on the platform
Get help	<code>%PLATFORM_HOME%/FRWK/bin/gemconnect help</code>	View help for the platform

The OSAGENT and TOMCAT Scripts

The **gemconnect** script runs several other scripts:

- **osagent**
- **tomcat**

You can run these scripts separately to start, stop, restart, or view the status of the OSAGENT or Tomcat Web servers independently.

LinqUs Provisioning Manager uses two Tomcat Web servers:

- A Tomcat Web server shared with all other plugged-in LinqUs Over-The-Air Suite products. Among other things, this server provides access to the GUI of LinqUs Provisioning Manager.
- A Tomcat Web server hosting the Web services component of LinqUs Provisioning Manager. This Tomcat 5.0.28 server is started and stopped by the scripts `startup.sh` and `shutdown.sh`, respectively. These scripts are located in the `webservices_tomcat_home/bin` folder.

The parameters of these scripts are the same as those of the **gemconnect** script:

start

Start the OSAGENT or Web server.

stop

Stop the OSAGENT or Web server.

restart

Stop and then start the OSAGENT or Web server.

state

Display the current status of the OSAGENT or Web server.

help

Display help.

Prerequisites

Before starting the OSAGENT, ensure it is configured to run on a valid CORBA port. The OSAGENT script in `%PLATFORM_HOME%/FRWK/bin` fails at startup if it cannot open a dedicated port (set in `gemconnect.env`). The **gemconnect** script fails at startup if it controls the OSAGENT and the OSAGENT itself cannot start.

Resetting LinqUs Provisioning Manager 4.1

To reset LinqUs Provisioning Manager:

- 1 Stop the LinqUs Provisioning Manager product using the Customer Care Interface (CCI).
- 2 Delete the provisioning command locks file:
rm -Rf \$PTF_HOME/PM/PM_instance/persistence/lock.db
- 3 Delete the batch file directories:
rm -Rf \$PTF_HOME/PM/PM_instance/batch_home
- 4 Remove all the invocations in LinqUs Provisioning Manager tables:
sqlplus LPM_DB_User/LPM_DB_Password@LPM_SID
truncate table see_invocation
- 5 Restart LinqUs Provisioning Manager.

Configuring User Access Rights

Before users can access the system, appropriate user accounts must be created and associated with user profiles. The relevant procedures are described in the *LinqUs OTA Manager 4.x Administration Guide*.

Users can only perform provisioning tasks if access to the **Provisioning Management** function has been specified in their user profile.

To specify the provisioning access right:

- 1 Log in to the CCI using an administrator account.
- 2 Either:
 - On the Welcome window, click **User Management**.
 - Click **User** on any management interface window.
- 3 Click **Profile** on the left-hand menu to display a list of profiles.
- 4 Select a profile and click **View** to display the properties of the profile for all installed LinqUs Over-The-Air Suite products:
- 5 Scroll down to the **List of Provisioning Manager functionalities** section and select **Provisioning Management**, as shown in “Figure 1”.

Figure 1 - User Access Rights

List of Service Manager functionalities			
<i>Database Repository</i>			
Management	<input checked="" type="checkbox"/>	Viewing	<input checked="" type="checkbox"/>
<i>Service Content</i>			
Provisioning	<input checked="" type="checkbox"/>		
<i>Invocation</i>			
Management	<input checked="" type="checkbox"/>		
<i>Campaign</i>			
Management	<input checked="" type="checkbox"/>		
List of Provisioning Manager functionalities			
<i>Provisioning</i>			
Management	<input checked="" type="checkbox"/>		
List of IMEI Manager functionalities			
<i>Repository</i>			
Management	<input checked="" type="checkbox"/>	Viewing	<input checked="" type="checkbox"/>
<i>Connector</i>			

6 Click **Update** to update the access rights.

Note: Each input connector is also assigned a user account. This user account must have an Administrator profile and be assigned **all** access rights to **all** targeted products.

Stopping an Incoming Connector's Input Flow

You can stop the flow of provisioning commands from a specified input connector (for example, the Web service API or the Core API input connector) using the CCI:

- 1 Click **Product Info > Input connectors** on the LinqUs Provisioning Manager main menu. The Status of Input Connectors window is displayed, showing a list of all currently available input connectors.
- 2 Activate or deactivate input connectors as required. Refer to the *Online Help* for details.

Stopping Provisioning of a Product

To stop provisioning commands being sent to a particular product:

- 1 Before stopping LinqUs Provisioning Manager, it is recommended to first stop the input connectors and allow all IN_PROCESS commands to complete processing. For example, to stop LinqUs Provisioning Manager during batch file processing, it is recommended to proceed as follows:
 - a) Stop the batch input connector by setting the CONNECTOR_BATCH/THROUGHPUT product parameter to 0—see “Modifying the Batchload Input Connector's Input Flow Rate” on page 13 for details.

- b) Wait for all “Executing” commands to complete (use the **Batch Load > View Current Activity** menu in the CCI).

IMPORTANT: When restarting LinqUs Provisioning Manager, do not forget to reset the input connector’s THROUGHPUT product parameter to its original value.

- 2 Stop LinqUs Provisioning Manager using the CCI.
- 3 Edit the `utility_connectors.properties` file (see “utility_connectors.properties” on page 43) and set the dedicated state properties to 0 (for example, `scm_connector_state = 0`). This deactivates the service connector. The service connector continues, however, to be listed on the Monitoring CCI page.
- 4 Restart LinqUs Provisioning Manager using the CCI.

Provisioning Recommendations

It is recommended that you:

- Only include one type of provisioning command in each batch file. Files containing more than one command—for example, **Create Subscription** and **Change MSISDN** commands targeting the same subscribers—would produce different results if stopped before completion then replayed from the beginning. Such files cannot therefore be replayed.
- Do not use the CCI interface to modify data that were created by batch file provisioning.

Deployment Considerations

For optimal performance, it is important to respect the hard disk mounting recommendations given in the *LinqUs Provisioning Manager 4.1 Installation Guide*.

LinqUs Provisioning Manager calls the Core API of targeted LinqUs Over-The-Air Suite products.

Correct configuration of the LinqUs Provisioning Manager core is important, in particular the **see.spj.threadPoolCapacity** parameter from the `tuning.properties` configuration file. See “tuning.properties” on page 36.

The value of the **see.spj.threadPoolCapacity** parameter must be less than the Visibroker layer’s internal thread pool size on each targeted LinqUs Over-The-Air Suite product. By default, this value is 20. If this recommendation is not followed, LinqUs Provisioning Manager may saturate the thread pool, resulting in a type of denial of service situation for the targeted product, particularly when a batch file is running.

You can add the following property to LinqUs Provisioning Manager’s CLASSPATH in order to increase the Visibroker thread pool size:

-Dvbroker.se.iioptp.scm.iioptp.dispatcher.threadMax=XX

where XX is the pool size.

Saturation can also occur when the Web Services API is used to submit commands: the number of threads submitted to LinqUs Provisioning Manager must be less than its Visibroker pool size.

It is recommended to deactivate the audit trail function. Its use can severely impact the performance of LinqUs Provisioning Manager.

Managing Databases

It is recommended that the indexes of the LinqUs Provisioning Manager database are analysed and rebuilt regularly (for example, every night at 12.00pm).

Batch File Processing

You can use batch files to execute multiple provisioning commands without manual intervention.

LinqUs Provisioning Manager 4.1 supports three types of batch files:

- Standard batch files for provisioning standard commands
- Card security and card content files (see “Direct Batch File Processing” on page 11).
- TBL batch files for provisioning TBL-compatible commands.

LinqUs Provisioning Manager 4.1 provides a set of dedicated directories in the server's file system into which standard and TBL batch files can be uploaded.

Batch files can be uploaded into these folders by:

- File transfer, for example using FTP. In this case, access to the batchload folder is controlled by standard operating system policies.
- Using the **Batch Load > Add file** page of the CCI interface. In this case, access is controlled by the Provisioning access right in the Framework user profile and batch file size is limited to 100MB.

Direct mode batch files cannot be uploaded using the CCI interface: they must be uploaded directly to designated folders on the server's file system.

Files deposited in the batchload directories are processed on a first-in, first-out basis, based on the file creation date. Individual commands within each file are then processed in reading order.

The speed with which individual batch commands are processed can be limited to a maximum execution throughput rate in order to preserve execution resources for higher priority requests arriving from other sources (for example, event processing or Web services). This maximum execution throughput rate is set at run time: refer to “Modifying the Batchload Input Connector's Input Flow Rate” on page 13 for details.

Similarly, a time frame for batch file processing can be specified. No batch file processing is then performed outside this time frame (though processing of running files continues). The execution time frame is similarly specified at run time: refer to “Modifying the Batchload Timeframe” on page 14.

A LinqUs Provisioning Manager 4.1 product instance can be stopped during batch file processing. In this case, batch file processing restarts from the exact point in the file that was being processing at the time that the product instance was stopped.

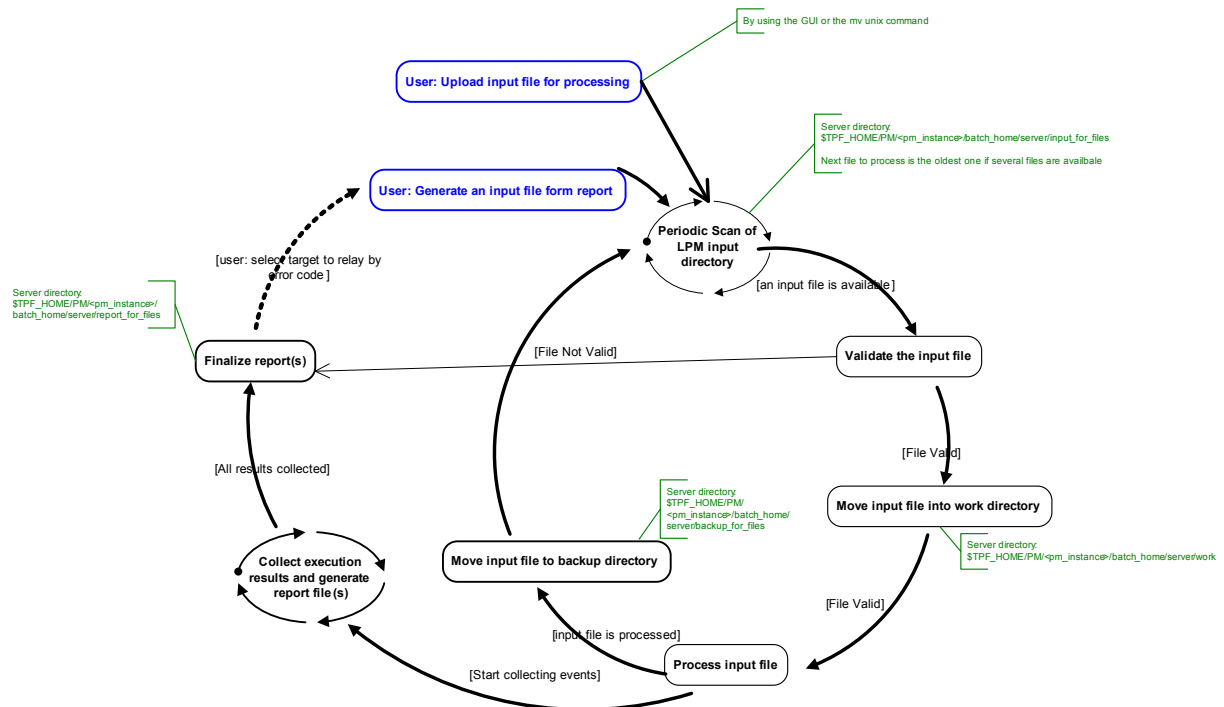
Warning: Information necessary for restarting batch file processing correctly is stored in the `ref` folder of each batch processing folder on the server. For example, the folder `$LPM_HOME/$INSTANCE/batch_home/server/input_for_files/ref` (where `$LPM_HOME/$INSTANCE` is the root installation folder of the LinqUs Provisioning Manager instance) contains restart information for input files. **Do not delete the files in these folders.**

Standard Batch File Processing

The Life Cycle of a Standard Batch Processing File

“Figure 2” illustrates the life cycle of a standard batch file from the time it is uploaded for processing through the CCI through to generation of a report file containing the results of processing.

Figure 2 - Batch Processing File Life Cycle



Standard Batch Processing File Format

Standard batch files are XML format files that reference an XML schema.

The XML schema (XSD) files are installed in the following folder:

`$LPM_HOME/$INSTANCE/config`

where `$LPM_HOME/$INSTANCE` is the root installation folder of the LinqUs Provisioning Manager instance.

The ProvisioningOrders Element

The root element of standard batch files is **ProvisioningOrders**. This element has the following attributes:

errorCodeFilterList Specifies a filter used to select the orders to execute. If an order in the file matches the filter, the order is executed. Otherwise, the order is ignored. The **errorCodeFilterList** attribute contains a string specifying a comma-separated list of error code filters.

The syntax of an error code filter is:

`[command:]errorCode[-subErrorCodeType[-subErrorCode]]`

where:

- *command* is the name of the provisioning command to match. For example, 'CreateSubscription'.
- *errorCode* is the LinqUs Provisioning Manager error code number to match. For example, '100' matches the error code PM-100 (refer to “Chapter 6 - Error Messages and Exceptions” for a complete list of error codes).
- *subErrorCodeType* is the error category returned by the product which provoke the error. For example 'RCA' is a category possibly returned by the Card Manager component of the OTA Manager product.
- *subErrorCode* is the error code returned by the product that provoked the error. For example '27' may be returned by the Card Manager (RCA) component of the OTA Manager whenever a request fails.

For example:

`UpdateSubScription:100,DeleteSubscription:1-RCA-27`

Using this filter, the only commands executed are **CreateSubscription** orders with a Provisioning Manager error code PM-100 (“Subscription does not exist on product”) and **DeleteSubscription** orders with error code PM-1 (“Unhandled error”) caused by an RCA error 27.

Note: Take care not to add any extra commas (“,”) to the **errorCodeFilterList** string, as an empty filter always matches no orders. In this case, the entire contents of the batch file are ignored.

generateReport Used to control batch report generation. If set to “false”, no report is generated by the batch execution. If set to “true”, a report is generated.

The Order Element

ProvisioningOrders elements contain one or more **Order** elements, each containing a single provisioning command. **Order** elements can have the following attributes:

endUser Can be set to indicate the “submitter” of this provisioning order. This information is retrieved in the monitoring of commands

transactionId Indicates the “transaction identifier” of this provisioning order. This information is retrieved in the monitoring of commands

internalId*	Unique identifier of the job that executed the order.
errorCode*	If an error occurs, the Provisioning Manager error code. See "Chapter 6 - Error Messages and Exceptions".
errorMessage*	A text message describing the error.
subErrorCodeType*	The error category returned by the product that provokes the error. For example 'RCA' is a category returned by the OTA Manager product. Refer to the <i>Administration Guide</i> of the product for details.
subErrorCode*	The error code returned by the product that provokes the error. For example '27' is returned by the Card Manager (RCA) component of the OTA Manager when a request fails.
subErrorMessage*	Text message describing the error returned by a product invoked by the provisioning command.
submissionDate*	The submission date of the order. Format is "yyyy/MM/dd HH:mm:ss:SSS"
endingDate*	The end date of the order. Format is "yyyy/MM/dd HH:mm:ss:SSS"
	Note: The submission and end dates are calculated for the machine on which the LinQUs Provisioning Manager product instance is running.

Note: The attribute values marked with an asterisk (*) are generated automatically by LinQUs Provisioning Manager during batch file processing and written to report files. They are subsequently used for processing the **errorCodeFilterList** attribute of the ProvisioningOrders element (see "The ProvisioningOrders Element" on page 8).

Examples

The following batch file contains a single **Change MSISDN** provisioning command:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<ProvisioningOrders xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xsi:schemaLocation="http://www.gemalto.com/schema/pm
pmBatchFile.xsd" xmlns="http://www.gemalto.com/schema/pm"
generateReport="true">
  <Order transactionId="Trans991111111">
    <ChangeMSISDN
      iccid      ="99992011110000000"
      imsi       ="88894922220000000"
      msisdn     ="9911111111"
      imei        ="359382005999999"
      cardProfile ="GXX_v3.2_128K_Copy"
      finalState  = "ACTIVE"/>
    </Order>
  </ProvisioningOrders>
```

The following example contains an **Update Subscription** provisioning command followed by service content contained within a **ServiceContent** element and an **ExecScript** element (calling the Generic Card Update 2G OTA Manager service) containing the contents of a script file, to be executed after the provisioning command completes:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<ProvisioningOrders xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xsi:schemaLocation="http://www.gemalto.com/schema/pm
pmBatchFile.xsd" xmlns="http://www.gemalto.com/schema/pm"
generateReport="true">
  <Order transactionId="Trans0612131313">
    <UpdateSubscription
      iccidSrc      ="994920111100000000"
      msisdnSrc     ="0612131313"
      imsi          ="008949222200000000"
      finalState    ="ACTIVE"
      imei          ="359382005999999"
      groupId       ="01"
      communicationProtocol ="SMS"
      serviceExecutionProtocol ="SMS">

    <ServiceContent>
      <Portal>
        <Final label="Post-Paid"
          majorVersion="1" minorVersion="0" />
      </Portal>
    </ServiceContent>

    <ExecScript>
      public class UpdateHPLMN2
      {
        public void script ()
        {
          // variable definitions
          short MF = 0x3F00;
          short DFGSM = 0x7F20;
          short HPLMN = 0x6F31;

          // Select HPLMN file
          selectById(MF);
          selectById(DFGSM);
          selectById(HPLMN);
          updateBinary((short)0, new byte[] {(byte)0x0B});
        }
      }
    </ExecScript>
  </UpdateSubscription>
</Order>
</ProvisioningOrders>
```

For information on the provisioning commands and their parameters, refer to the *LinqUs Provisioning Manager 4.1 Command Parameters Guide* and the sample files provided.

Standard Batch Report File Format

When LinqUs Provisioning Manager 4.1 finishes processing all commands in a batch file, the results are written to a XML report file in the directory `/batch_home/server/work` in the standard batch file processing format.

Report files are named:

originalfilename_Report_yyyymmdd_hhmmss

where *yyymmdd_hhmmss* represents the creation date of the report file.

For each processed input file a CDR record is logged using Framework Billing module. The format of the CDR record is described in “Table 4 - Standard CDR Record Format” on page 20.

When file processing is complete, the report file is moved to the `backup` folder.

Replayed Report File Format

The format of a replayed report file is:

originalFileName_report_originalDate_replay_index_report_replayDate

where:

- *originalDate* represents the creation date of the original report file
- *replayDate* represents the date of the creation of the report resulting from the replay.
- *index* indicates the level of recursivity.

Both *originalDate* and *replayDate* use the *yyymmdd_hhmmss* format.

For example, if you replay the initial report

`BATCH_CreateSubscription_report_20091103_105521`, the name of the new report resulting of the replay might be, depending on the date it is replayed:

`BATCH_CreateSubscription_report_20091103_105521_replay_1_report_20091103_105601`

It is also possible to replay a report that is already the result of a previous replay. In this case, the value of the *index* parameter is incremented. For example, if you replay the report resulting from replaying the initial report, the name of the new report might be, depending on the date it is replayed:

`BATCH_CreateSubscription_report_20091103_105601_replay_2_report_20091103_105743`

The *index* parameter indicates the level of recursivity of the replay with respect to the initial report file. In this example, an *index* value of 2 indicates a “replay of a replay”.

Direct Batch File Processing

OTA Manager batch files (Card Content and Card Security files) can be automatically processed by the LinqUs Provisioning Manager’s Direct Batch File connector.

In Direct batch mode, the contents of the batch file are not interpreted by LinqUs Provisioning Manager in any way, but passed directly to OTA Manager.

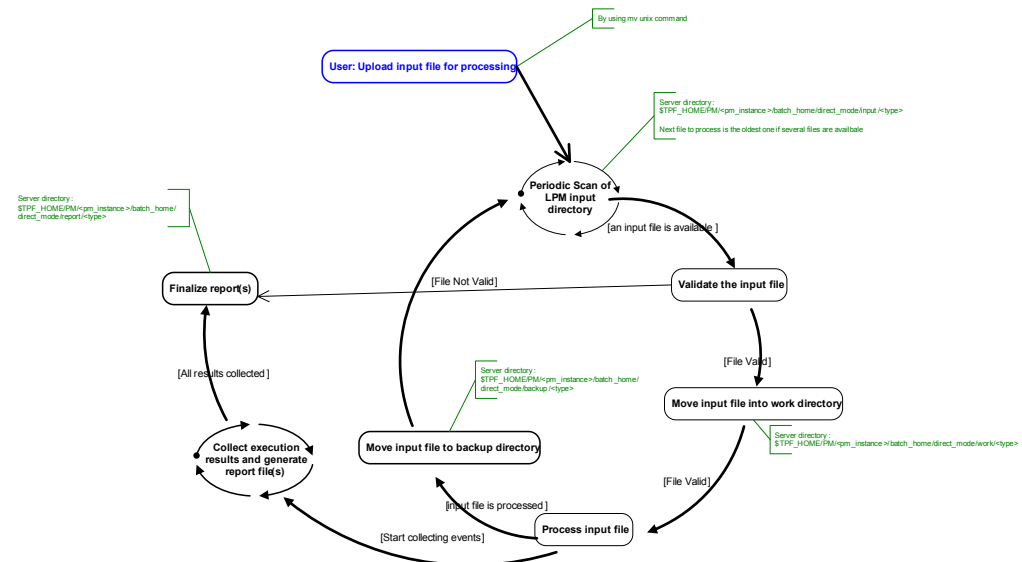
Note: It is not recommended to use Direct mode for Card Content. Unlike standard mode, Direct mode can only be used to provision OTA Manager, does not generate detailed reports on the cards created or updated, and does not allow batch files to be replayed.

Direct mode is used to upload card content and card security files. The commands **Upload Card Content File** and **Upload Card Security File** are available for this purpose.

The Life Cycle of a Direct Mode File

“Figure 3” illustrates the life cycle of a Direct mode batch file.

Figure 3 - Direct Mode File Life Cycle



Direct Batch File Format

The following types of direct batch files can be processed:

- Card Security Files
- Card Content Files

Card Security Files

Card security files are XML files that must conform to the `messageBuilder.dtd` defined by the OTA Manager product. This file is listed in the *OTA Manager 4.x Administration Guide*.

Card Content Files

Card content files are XML files that must conform to the `cardframework.dtd`. This file is listed in the *OTA Manager 4.x Administration Guide*. Card security files accompanying card content files must conform to the `messageBuilderV2.dtd`.

Direct Mode Folders

Card content and card security files for direct transfer to OTA Manager must be uploaded into specific folders:

- `batch_home/direct_mode/input/content/create`
Contains card content files to be created in the OTA Manager database.
- `batch_home/direct_mode/input/content/update`
Contains card content files to be updated in the OTA Manager database.
- `batch_home/direct_mode/input/security/create`
Contains card security files to be created in the OTA Manager database.

- `batch_home/direct_mode/input/security/update`

Contains card security files to be updated in the OTA Manager database.

When LinqUs Provisioning Manager has finished processing a file, it is copied to a corresponding backup folder, for example `batch_home/direct_mode/backup/security/update`.

A CDR record is logged for each error that occurs while processing input files.

Direct Mode Report File Format

For each file processed in direct mode, a report file is generated in a `report` folder structure corresponding to the input file folder structure, for example `batch_home/direct_mode/report/content/create`.

Report files are named as follows:

Report_yyyymmdd_hhmmss.txt

where `yyyymmdd_hhmmss` represents the creation date of the report file.

Each line of the report file is as follows:

```
11/09/2007 04:06:02:785;;com.gemplus.gse.comp;FAILED;[ICCID
'80000000000000000009'] error: ObjectNotFoundException: [errorCode:4]
Profile '04.50' does not exist in repository.
```

Where:

Table 2 - Direct Mode Report File Format

Field	Description
1	Date and time of the command's execution.
2	A blank field.
3	Mode.
4	Status (only failed commands are written into the report)
5	Error message returned by OTA Manager.

Maintaining Batch Files

To prevent the possibility of overflowing the server's file system, you should schedule an administrative task to regularly delete:

- All batch files contained in the `backup_for_files` and `backup` folders.
- All batch files contained in the `report_for_files` and `report` folders that are not currently being processed.

Monitoring Batch File Commands

All commands issued from within a batch file can be monitored using the CCI. See "Chapter 2 - Monitoring".

Modifying the Batchload Input Connector's Input Flow Rate

You can modify the flow of provisioning commands from the Batchload input connector without restarting the LinqUs Provisioning Manager product:

- 1 Access the Product Parameters page in the CCI, as described in "Changing Product Parameter Values" on page 33.

- 2 Set the value of the CONNECTOR_BATCH/THROUGHPUT product parameter:
 - Specify 0 to stop the incoming command flow completely.
 - Specify -1 to use the maximum available throughput.
 - Specify an integer value as the throughput rate in terms of commands per second.


See “Product Parameters” on page 33 for a complete list of product parameters.

Modifying the Batchload Timeframe

There are two possibilities:

- 1 Modify the value of the CONNECTOR_BATCH/TIME_FRAME product parameter in the CCI, using the procedure described in “Changing Product Parameter Values” on page 33.

This product parameter can be “hot” modified and the new value is applied on the next product restart.

- 2 Modify the daily time frame value directly in the CCI as follows:
 - a) Click **Product Info > Input connectors** on the LinqUs Provisioning Manager main menu. The Status of Input Connectors window is displayed.
 - b) Specify the **Daily time frame**. Click the time control  and select new values for the start or end time. Refer to the *Online Help* for details.

Managing Locks

Provisioning commands can be executed with a “guarantee of execution”, meaning that if a product instance is stopped (or crashes) and then restarted, all the commands that were in progress are automatically re-executed from the beginning when the product instance is restarted.

Lock management means that the “lock order” of provisioning commands must be respected to allow the commands to be re-executed in the same order that they were originally received in.

The lock is executed on the MSISDN, ICCID, or IMSI identifiers.

Note: When restarting LinqUs Provisioning Manager after an unexpected stop (crash or process kill), unprocessed commands are played again. Some commands may appear to be waiting for a lock (when viewed in the Monitoring window of the CCI). Unfortunately, this lock may never be acquired because LinqUs Provisioning Manager stopped before being capable to register them with the lock manager.

The recommended approach is to let these commands expire and replay them later.

The Lock File

Locks are stored permanently in a dedicated file located under the %PM_INSTANCE_HOME%/persistence directory. The file name is lock.db.

The persistence directory is automatically checked at LinqUs Provisioning Manager product startup. If the directory does not exist it is created.

The lock file is also automatically checked at LinqUs Provisioning Manager product startup. If it does not exist, it is automatically created.

To avoid having an infinitely growing file (and consequent “out of memory” exceptions), the lock manager allows only:

- A maximum numbers of locks
- A maximum number of commands allowed to wait for a lock.

These two tuning parameters are defined in the `tuning.properties` file (see “tuning.properties” on page 36).

Within the following constraints, the file may be a simple indexed file:

- Each record stores an instance of `LockedTarget`.
- The size of `LockedTarget` is predefined (since the number of commands allowed to wait for a lock is limited)
- The size of the file is predefined by the formula:
*recordSize * max_number_of_locks*

When Locks Are Stored

Locks are permanently stored in the following circumstances:

- When a lock is created in the lock repository, the **CommandLockPersistenceManager** is asked to save the new entry. It may detect it is a new entry with the `LockedTarget.index` attribute value that should be -1. When saved, the `LocketTarget` instance should have an index attribute value not equal to -1.
- When the lock waiter list of a `LockedTarget` gets empty, the **CommandLockPersistenceManager** is asked to remove the entry from the file. In this case the `LocketTarget.index` indicates the location of this entry in the file. The entry gets available for a new target.

The record position in the file may be computed with the following formula:

*target_index * record_size*

Command Lock Manager Configuration

The command lock manager offers some tuning parameters. These parameters are defined in the `tuning.properties` file of each LinqUs Provisioning Manager instance.

See “tuning.properties” on page 36 for details.

Monitoring

You can monitor the current status or history of all provisioning command received by LinqUs Provisioning Manager 4.1

You can monitor provisioning commands using:

- The Customer Care Interface (CCI)
- A suitable SNMP console.
- The SuMoS product, if the package is used.

You can monitor:

- Commands currently executing. You can monitor the current state of execution advancement

During execution you can monitor, for example:

- The input connector name: the name of the input connector that requested the command's execution
- The end user name: depending on the input connector, the name of the user that requested the command's execution. For example, the Web service API or Core API input connectors both supply this information.
- The transaction identifier: the information received or set by the input connector.
- The invocation identifier: the key field set in the Provisioning Manager database.

- Completed commands. You can examine the history of the command's execution and the result of the command.

For each executed action, you can obtain the execution date, the global status, whether the command succeeded or failed, and a descriptive message.

This is an example of the history of a **Create Subscription** command:

```
CreateSecurity    OK    Security was already defined for this card
CreateCard       OK    Card successfully created
ApplyPortal      KO    No service profile defined
```

Each provisioning request monitored is accompanied a full history allowing analysis of all the tasks performed by the request: "skipped" if an action was bypassed, "verified" or "unverified" to indicate whether a statechart transition was valid or not.

Provisioning request that result in errors are accompanied by an error code with details of the error that occurred. See “Chapter 6 - Error Messages and Exceptions” for a complete list of error codes. Error codes are classified as follows:

- 1 to 10,000. Indicates a fatal error. The provisioning command has failed because its execution parameters that are not consistent with the targeted platform data. In this case there is no point in replaying the original command.
- 10001 and up. Indicates a temporary error. The command may be replayed without updating its parameters. However, manual intervention may be required before replaying the command (for example: activating a service connector)

Log File, Audit Trail and Billing Ticket Formats

The Provisioning Manager generates its own log file, audit trail, and billing records.

Consulting the Log File, Audit Trail and Billing Tickets

Logging

The logging facility provides a history of product activity. Logging can also be used for debugging in both development and production environments.

The Provisioning Manager Customer Care Interface (CCI) allows you to configure the log file (**Platform > Platform configuration > Logging**).

Audit Trail

The Audit Trail facility provides services for generating a trace of the actions performed by the users of the system.

The Provisioning Manager CCI enables you to configure the audit trail (**Platform > Platform configuration > Audit trail**).

Format of the Log File

Log file entries are generated by the Framework. Refer to the *OTA Manager 4.x Administration Guide* for details.

Format of the Audit Trail File

Audit trail entries are generated by the Framework. Refer to the *OTA Manager 4.x Administration Guide* for details.

Format of the Billing Ticket Files

Command Execution CDR Record Format

For each executed provisioning command, a Call Detail Record (CDR) is written to the Framework's Billing module. Each CDR record provides the following information:

Table 3 - Billing Ticket Fields

Billing Ticket Parameter	Description
Record Number	Number of the billing ticket in the current billing file.
Product Instance Name	Product instance name (for example, "LPM1").
Timestamp	The provisioning transaction time according to Framework, in YYYYMMDDHHMMSS format.
Request ID	The internal provisioning command identifier
Transaction ID	Text (Transaction ID) used to identify the provisioning transaction, if any. Otherwise blank.
Ticket type	Value is always "PROVISIONING_CMD" for command execution.
User ID	User identifier of the provisioning command, if any. Otherwise blank (a command issued from a batch file has no user).
Connector name (invoker)	Name of the connector that submitted the command.
Beneficiary	This field is empty.
Command name	The name of the command (same as that seen in the CCI)
End status	
Error code	EXPIRED, SUCCEEDED or FAILED
MSISDN	Mobile Station ISDN Number. Mobile phone number that received the provisioning transaction. May be empty if an ICCID or IMSI are specified.
ICCID	Integrated Circuit Card Identifier (ICCID). May be empty if MSISDN or IMSI are specified.
IMSI	International Mobile Subscriber Identifier (IMSI). May be empty if MSISDN or ICCID are specified.

Batch File Connector CDR Record Format

For each processed standard batch file, a CDR record is logged to the Framework's Billing module. The format of each record is as follows:

Table 4 - Standard CDR Record Format

Name	Description
Record number	Number of the billing ticket in the current billing file.
Product instance name	Product instance name (for example, "LPM1").
Timestamp	The provisioning transaction time according to the Framework, in YYYYMMDDHHMMSS format.
Request Id	Blank.

Table 4 - Standard CDR Record Format (continued)

Name	Description
Transaction Id	Blank
Ticket type	Value is always "PROVISIONING_FILE" for input file processing.
Connector name	Name of the connector that processed the file.
File name	Name of the processed file.
Number of commands	Number of processed commands in the file.

Direct File Connector CDR Record Format

For each processed direct file, a CDR record is logged to the Framework's Billing module. The format of each record is as follows:

Table 5 - Direct File Connector CDR Record Format

Name	Description
Record number	Number of the billing ticket in the current billing file.
Product instance name	Product instance name (for example, "LPM1").
Timestamp	The provisioning transaction time according to Framework, in YYYYMMDDHHMMSS format.
Request ID	Blank
Transaction ID	Blank
Ticket type	Value is always "DIRECT_FILE" for direct file processing.
End User ID	Blank
Connector name	Name of the connector that processed the file
Beneficiary	Blank
File name	Name of the processed file
File type	Type of processed file: 'CardContent' or 'CardSecurity'
Action	Type of operation: 'Create', 'Update', or 'Delete'

SNMP Traps and Counters

The Simple Network Management Protocol (SNMP) facility in Provisioning Manager can be accessed as an SNMP agent from a network management platform using the SNMP V1 protocol. In this way, Provisioning Manager can give notice of problems and generate and send alarms to a management platform using the SNMP trap format.

The SNMP facility is configured through the management interface. You can enable or disable the trap function and specify the address of the network management platform using the Provisioning Manager graphical user interface (GUI) (**Platform > Platform config.> SNMP**).

The MIB File

All items of information about LinqUs Provisioning Manager 4.1 that can be monitored using SNMP are described in a hierarchical file called a Management Information Base (MIB).

An SNMP Manager can request and collect information from an SNMP agent's MIB file, as well as inspect the objects contained therein. For example, from the SNMP Manager you can examine the number of times the product has been started and stopped.

Each element manages specific objects, with each object having specific characteristics. Each object and characteristic has a unique object identifier (OID) consisting of numbers separated by decimal points (for example, 1.3.6.1.4.1.5496.2.20). These object identifiers naturally form a hierarchical tree structure. The MIB associates each OID with a readable label and various other parameters related to the object.

Here is an example of one of the OIDs used by Provisioning Manager (5496.2.20):

Table 6 - An Example OID

Code	Value
5496	Gemalto node
2	Provisioning Manager node
20	Platform status node

SNMP Traps

Traps are generated by either Framework facilities (billing, logging, audit trail) or Provisioning Manager product components.

Each trap contains the following information:

- Code: uniquely identifies the alarm
- Type: classifies the alarm domain
- Timestamp.

Note that no severity escalation is employed: one trap is generated for notification of service unavailability and another trap for service availability.

In addition, each alarm can contain the following information as variable bindings:

- Sender name: the module or product that detects the alarm
- Host name: the name of the host machine
- ORB agent port (for facilities alarms, this uniquely identifies the platform)
- Driver identifier for channel alarms
- Exception name: the name of the exception that raised the alarm
- Contextual message: information to help analyze and solve the problem; this message may be empty if previous fields contain all the necessary information.

The following tables provide information on the possible traps.

Provisioning Manager Traps

Trap Name	Cause	Variable Binding
instanceStop	A Provisioning Manager instance is stopped	Sender name (facility name) Host name ORB agent port (uniquely identifies the Framework if two platforms are running on the same host) Exception name Contextual message (error message)
instanceUp	A Provisioning Manager instance is started	Sender name (facility name) Host name ORB agent port (uniquely identifies the framework if two platforms are running on the same host) Contextual message (error message)
databaseCnxDown	Database connection lost	Sender name (facility name) Host name ORB agent port (uniquely identifies the framework if two platforms are running on the same host) Exception name Contextual message (error message)
databaseCnxUp	Database connection restored	Sender name (facility name) Host name ORB agent port (uniquely identifies the framework if two platforms are running on the same host) Contextual message (error message)
utilityCnxDown	A service connector lost the connection to its associated product (only for the ACTIVATED connector)	Sender name (facility name) Host name ORB agent port (uniquely identifies the framework if two platforms are running on the same host) Exception name Contextual message (name of the connector that lost the connection)
utilityCnxUp	A service connector restores the connection to its associated product (only for the ACTIVATED connector).	Sender name (facility name) Host name ORB agent port (uniquely identifies the framework if two platforms are running on the same host) Contextual message (name of the connector that restored the connection)
inputCntrActivated	An Input connector is activated	Sender name (facility name) Host name ORB agent port (uniquely identifies the framework if two platforms are running on the same host) Exception name Contextual message (name of the connector activated)
inputCntrDeactivated	An input connector is deactivated	Sender name (facility name) Host name ORB agent port (uniquely identifies the framework if two platforms are running on the same host) Contextual message (name of the connector deactivated)

Trap Name	Cause	Variable Binding
inputCntrFrozen	An input connector is frozen	Sender name (facility name) Host name ORB agent port (uniquely identifies the framework if two platforms are running on the same host) Contextual message (name of the connector frozen)
pmSEEAAlert	Alert from internal Service Execution Engine of Provisioning Manager server	Sender name(facility name) Host name ORB agent port (uniquely identifies the framework if two platforms are running on the same host) Component ID Exception name Contextual message (error message)
pmSEEAAlertCanceled	Alert canceled from internal Service Execution Engine of Provisioning Manager server	Sender name(facility name) Host name ORB agent port (uniquely identifies the framework if two platforms are running on the same host) Component ID Exception name Contextual message (error message)
mcieCommandPostFailed	Change IMEI Event command post failed	Sender name (facility name) Host name ORB agent port (uniquely identifies the framework if two platforms are running on the same host) Contextual message (error message)

SNMP Counters

Platform Status

When an SNMP manager is launched (after Provisioning Manager is started), the following persistent traps provide information about the platform. Each time a trap is sent, the corresponding status value is updated. The value is set to 1 if the service is functioning correctly, and to 0 if a warning trap (down or trouble) was sent.

Table 7 - Persistent Traps for the Platform

Name	Type	Description	Trap
localBilling	Integer	Status of local billing module in File mode	localBilling Up/Down
storageBilling	Integer	Status of local billing module in Storage mode	storageBilling Up/Down
remoteBilling	Integer	Status of remote billing module	remoteBilling Up/Down
localLogging	Integer	Status of local logging module in File mode	localLogging Up/Down
storageLogging	Integer	Status of local logging module in Storage mode	storageLogging Up/Down
remoteLogging	Integer	Status of remote logging module	remoteLogging Up/Down
localAuditTrail	Integer	Status of local audit trail module in File mode	localAuditTrail Up/Down

Table 7 - Persistent Traps for the Platform (continued)

Name	Type	Description	Trap
storageAuditTrail	Integer	Status of local audit trail module in Storage mode	storageAuditTrail Up/Down
remoteAuditTrail	Integer	Status of remote audit trail module	remoteAuditTrail Up/Down
auditTrailDatabase	Integer	Status of audit trail module in database mode	auditTrailDatabase Up/Down

Provisioning Manager Counters

For each product instance, Provisioning Manager provides statistical information through the SNMP protocol. This information includes single values and the following tables:

- **pmTable**
- **icWebSrvTable**
- **icBatchLoadTable**
- **icDirectTable**
- **icDDMEventsTable**
- **icPBEventsTable**
- **ucOTATable**
- **ucSMTTable**
- **ucDMTable**
- **ucSCMTable**
- **ucPBTable**

pmTable

The **pmTable** (index **pmIndex**) provides the information about Provisioning Manager instances.

Table 8 - pmTable Statistics

Value Name	Value Type	Description
pmOrbAgentPort	Integer	Platform to which the product is connected
pmAddress	DisplayString	Provisioning Manager instance address
pmName	DisplayString	Provisioning Manager instance name
pmDatabase	Integer	Status of the database connection (1 = OK)
pmMemoryTotal	Integer	Total memory of the process
pmMemoryFree	Integer	Amount of free memory
pmNbInputConnector	Integer	Number of deployed input connectors
pmNbUtilityConnector	Integer	Number of deployed service connectors
pmUpTime	TimeTicks	Up time of this Provisioning Manager instance

icWebSrvTable

The **icWebSrvTable** (index **icwsPmId**) provides information about standard Web services commands. One table exists for each instance of the Provisioning Manager.

Table 9 - icWebSrvTable Statistics

Value Name	Value Type	Description
icwsName	DisplayString	Input connector name
icwsState	Integer	Input connector state (1=ACTIVATED)
icwsNbOfCommands	Integer	Number of commands issued from this input connector
icwsThroughput	Integer	Current command throughput rate
icwsRunningTime	TimeTicks	Total time input connector has been running since startup (includes only time the connector is in ACTIVATED state).
icwsResultThroughput	Integer	Current command result throughput rate

icBatchLoadTable

The **icBatchLoadTable** (index **icblPmId**) provides information about standard Batch commands. One table exists for each instance of the Provisioning Manager.

Table 10 - icBatchLoadTable Statistics

Value Name	Value Type	Description
icblName	DisplayString	Input connector name.
icblState	Integer	Input connector state (1=ACTIVATED).
icblNbOfCommands	Integer	Number of commands issued from this input connector.
icblThroughput	Integer	Current command throughput rate.
icblRunningTime	TimeTicks	Total time input connector has been running since startup (includes only time the connector is in ACTIVATED state).
icblRequestedThroughput	Integer	Requested command throughput state.
icblResultThroughput	Integer	Command result throughput rate.
icblCurrentInputFile	DisplayString	Name of the currently processed file.
icblStartingDate	DisplayString	Processing start date of the current file.
icblNbCommandsTotal	Integer	Number of commands in the file currently being processed.
icblNbCommandsAccepted	Integer	Number of accepted commands in the file currently being processed.
icblCurrentReportFile	DisplayString	Name of the currently processed report file.
icblNbCommandsSucceeded	Integer	Number of successfully executed commands in the file currently being processed.
icblNbCommandsFailed	Integer	Number of executed commands in error in the file currently being processed.

icDirectTable

The **icDirectTable** (index **icdIPmId**) provides information about the direct batch processor that is deleted to Provisioning Manager. One table exists for each instance of the Provisioning Manager.

Table 11 - icDirectTable Statistics

Value Name	Value Type	Description
icdIName	DisplayString	Input connector name
icdIState	Integer	Input connector state (1=ACTIVATED)
icdINbOfCommands	Integer	Number of commands issued from this input connector
icdIThroughput	Integer	Current command throughput rate
icdIRunningTime	TimeTicks	Total time input connector has been running since startup (includes only time the connector is in ACTIVATED state).

icDDMEventsTable

The **icDDMEventsTable** (index **icddmPmId**) provides information about Device Detection events for the Input connector. One table exists for each instance of the Provisioning Manager.

Table 12 - icDDMEventsTable Statistics

Value Name	Value Type	Description
icddmName	DisplayString	Input connector name
icddmState	Integer	Input connector state (1=ACTIVATED)
icddmNbOfCommands	Integer	Number of commands issued from this connector
icddmThroughput	Integer	Current command throughput rate
icddmRunningTime	TimeTicks	Total time input connector has been running since startup (includes only time the connector is in the ACTIVATED state).
icddmChangeEvent	Integer	Number of received change events
icddmInfoEvents	Integer	Number of received info events
icddmRecoverEvents	Integer	Number of received recover events
icddmFailedPosts	Integer	Number of failed MCIE command posts

icPBEventsTable

The **icPBEventsTable** (index **icpbPmId**) provides information about Phone Book events for the input connector. One table exists for each instance of the Provisioning Manager.

Table 13 - icPBEventsTable Statistics

Value Name	Value Type	Description
icpbName	DisplayString	Input connector name
icpbState	Integer	Input connector state (1=ACTIVATED)
icpbNbOfCommands	Integer	Number of commands issued from this input connector
icpbThroughput	Integer	Current command throughput rate
icpbRunningTime	TimeTicks	Total time input connector has been running since startup (includes only time the connector is in the ACTIVATED state).

ucOTATable

The **ucOTATable** (index **ucotaPmId**) provides information about the OTA Manager service connector's status. One table exists for each instance of the Provisioning Manager.

Table 14 - ucOTATable Statistics

Value Name	Value Type	Description
ucotaName	DisplayString	Service connector name
ucotaState	Integer	Service connector state (1=ACTIVATED)
ucotaStatus	Integer	Service connector status (1=CONNECTED)
ucotaThroughput	Integer	Current connector throughput rate.

ucSMtable

The **ucSMTable** (index **ucsmPmId**) provides information about the Service Manager service connector's status. One table exists for each instance of the Provisioning Manager.

Table 15 - ucSMTable Statistics

Value Name	Value Type	Description
ucsmName	DisplayString	Service connector name
ucsmState	Integer	Service connector state (1=ACTIVATED)
ucsmStatus	Integer	Service connector status (1=CONNECTED)
ucsmCreatedServiceContents	Integer	Number of created service contents
ucsmServiceContentCreationThroughput	Integer	Service content creation throughput
ucsmInvocationCalls	Integer	Number of calls to Invocation layer
ucsmInvocationThroughput	Integer	Invocation calls throughput
ucsmResultThroughput	Integer	Invocation results throughput

Table 15 - ucSMTable Statistics (continued)

Value Name	Value Type	Description
ucsmSucceededResults	Integer	Number of successful invocation results
ucsmRejectedResults	Integer	Number of rejected invocation results
ucsmAbandonedResults	Integer	Number of abandoned invocation results
ucsmExpiredResults	Integer	Number of expired invocation results
ucsmBusyResults	Integer	Number of busy invocation results
ucsmDeletedResults	Integer	Number of deleted invocation results
ucsmSuppressedResults	Integer	Number of suppressed invocation results
ucsmFailedResults	Integer	Number of failed invocation results
ucsmLockedResults	Integer	Number of locked invocation results

ucDMTable

The **ucDMTable** (index **ucdmPmId**) provides information about the Device Manager service connector's status. One table exists for each instance of the Provisioning Manager.

Table 16 - ucDMTable Statistics

Value Name	Value Type	Description
ucdmName	DisplayString	Service connector name
ucdmState	Integer	Service connector state (1=ACTIVATED)
ucdmStatus	Integer	Service connector status (1=CONNECTED)
ucdmCreatedHandsets	Integer	Number of created handsets
ucdmHandsetCreationThroughput	Integer	Current throughput of handset creation
ucdmDeletedHandsets	Integer	Number of deleted handsets
ucdmHandsetDeletionThroughput	Integer	Current throughput of handset deletion

ucSCMTable

The **ucSCMTable** (index **ucscmPmId**) provides information about the Subscriber Manager service connector's status. One table exists for each instance of the Provisioning Manager.

Table 17 - ucSCMTable Statistics

Value Name	Value Type	Description
ucscmName	DisplayString	Service connector name
ucscmState	Integer	Service connector state (1=ACTIVATED)
ucscmStatus	Integer	Service connector status (1=CONNECTED)

Table 17 - ucSCMTable Statistics (continued)

Value Name	Value Type	Description
ucscmCreatedSubscribers	Integer	Number of created subscribers
ucscmSubscriberCreationThroughput	Integer	Current subscriber creation throughput
ucscmDeletedSubscribers	Integer	Number of deleted subscribers
ucscmSubscriberDeletionThroughput	Integer	Current subscriber deletion throughput
ucscmLoadedSubscribers	Integer	Number of loaded subscribers
ucscmUpdatedIMEIs	Integer	Number of updated subscriber IMEIs
ucscmUpdatedIMSI	Integer	Number of updated IMSIs

ucPBTable

The **ucPBTable** (index **ucPBPMId**) provides information about the Phonebook Backup service connector's status. One table exists for each instance of the Provisioning Manager.

Table 18 - ucPBTable Statistics

Value Name	Value Type	Description
ucPBName	DisplayString	Service connector name
ucPBState	Integer	Service connector state (1=ACTIVATED)
ucPBStatus	Integer	Service connector status (1=CONNECTED)
ucPBThroughput	Integer	Current throughput rate of the service connector.

Product Configuration

This chapter lists the product parameters for LinqUs Provisioning Manager 4.1, and describes the product's configuration parameters.

Product Parameters

The following tables describe the product parameters for each component of Provisioning Manager, and give the default value of each parameter.

Changing Product Parameter Values

When a Provisioning Manager product instance is created, the default product parameter values are stored in the product instance's database. Thereafter, the parameter values can be changed through the management interface; the new values are stored in the database and used for all future operations.

All Provisioning Manager parameters are “hot start”, that is, modified values are used immediately and do not require a product restart.

To modify a product parameter value, proceed as follows:

- 1 Click **Platform** in the top menu.
- 2 Click **Product config** on the left-hand menu.
- 3 Select the product instance in the product list and click **Configure**.
- 4 From the Product Configuration window, click the **Edit Parameters** radio button.
- 5 In the Product Parameters List window, modify values as required, then click **Update List**.

Note that when you change a parameter value, it is changed for only the selected product instance. You can use the export/import facility to copy changes to other product instances.

Provisioning Manager Product Parameters

Table 19 - Provisioning Manager Product Parameters

Parameter Group	Parameter Name	Parameter Description and Default Value	Hot/Cold
CONNECTOR_BATCH	TIME_FRAME	The running time frame of the input connector. This parameter is optional: it may be missing or undefined. In this case no time frame is defined. Format is: <i>hh:mm-hh:mm</i> . The start time must be before the end time. Default: -	H
CONNECTOR_BATCH	THROUGHPUT	The throughput rate, in commands/second, of the input connector. This parameter is optional: it may be missing or undefined. In this case, the default value is used. The value -1 means "no limit". Default: 10	H
DEFAULT_VALUE	EXEC_SCRIPT_MODE	The default mode for sending Exec script service execution requests to OTA Manager. Possible values are "2G" and "3G". Default: 2G	H
DEFAULT_VALUE	OTA_PROTOCOL	Indicates the default OTA protocol for requesting an OTA service execution. Accepted values are "SMS" and "CAT-TP". Default: SMS	H
DEFAULT_VALUE	COMMAND_VP	The default validity period for command execution, in minutes. The value -1 specifies an infinite validity period. Default: 1440 (1 day)	H
DEFAULT_VALUE	SERVICE_VP	The default validity period for product invocations, in minutes. The value -1 specifies an infinite validity period. Default: 4320 (3 days)	H
DEFAULT_VALUE	SM_PROTOCOL	The default Service Manager protocol to use. Default: SMS	H
DEFAULT_VALUE	SM_PERSO_PORTAL	The default personalization portal used in commands when the personalization portal is not specified.	H
COMMAND	DELETE_SUB_HANDLE_PBM	Indicates whether the provisioning command DeleteSubscription removes the specified card from the Phonebook Backup. Accepted values: true / false Default: False	H
COMMAND	MCIE_CARD_CHECK_DELAY	The default delay by the ManageDDMChangelmeiEvent command between two card existence checks, in minutes. Default: 60	H
COMMAND	MCIE_PB_CHECK_DELAY	The default delay by the ManageDDMChangelmeiEvent provisioning command between two autoregistration command end checks, in minutes. Default: 10	H

Configuration Parameters

config.properties

Displays a list of configuration parameters and their current values in the config.properties file.

```
# -----
# Configuration parameter values
# -----
#
# - ${<group>,<name>,<default>} will be replaced by the corresponding
product
# parameter <group>/<name> value (if the parameter does not exist, the
# <default> value will be used)
# ex:
# jobProcessorThreadPoolCapacity = ${JOB_PROCESSOR,NB_THREADS,20}
#
# - formula can be used with previously defined parameters
# ex:
# jobProcessorEventSchedulerPeriod = 20
# jobProcessorEventSchedulerSSAMemorySize = 3 * 24 * (3600 /
jobProcessorEventSchedulerPeriod)

# -----
# General configuration parameters
# -----

#Contains the root dir of the PM instance
#type: String, ex:/product/linqus/PM/LPM_VALID
#remark: mandatory
pm.instance.location =/product/linqus/PM/PM

#Contains the directory where are localized the custom jars
#The jars contained in this directory are added to the classPath ONLY of
# input connectors, utility connectors and provisioning commands
pm.customPluginsJars.location=/product/linqus/PM/lib/custom

# Contains the regular expression with which the properties' values
will be verified
# type: String, example: [A-Za-z0-9:._@/_]+ or [^<>]*
# remark: optional.
# The properties' values of all files of config/ directory are checked
# No verification is done on properties files of config/custom/
directory
# If this parameter does not exist or is empty, no verification is done.
pm.param.regularExpr = [^<>]*

#Contains the directory where are localized the scripts executed by the
TBLExecScript2G command
#type: String, ex:/product/linqus/PM/shared/execScript
#remark: mandatory
pm.tbl.otaScriptsLocation =/product/linqus/PM/shared/execScript

# -----
# Command configuration parameters
# -----
```

```
#Parameter to define a list of commands to not publish
#type: comma-separated strings list,
ex:TBLBasicChangeMSISDN,TBLChangeSIMCard,TBLCreateCardInfo
#remark: optional
pm.commands.unpublished =

#Parameter to define the state for the source card in case of changecard
command
#type: String
#remark: mandatory
tblcommand.changecard.srccardstate = INACTIVE

#Parameter to define the change number command mode (either BASIC or
SMART)
#type: String
#remark: mandatory
tblcommand.changemsisdn.mode = BASIC

# -----
# Parameters for see database connectivity. These values are used for
# all various SEE connections
# -----
#database url, jdbc format
#type: String, ex:jdbc:oracle:thin:@G099170:1521:lpn
#remark: mandatory
see.dbUrl = jdbc:oracle:oci:@LUDB

#jdbc driver class name
#type: String, ex:oracle.jdbc.driver.OracleDriver
#remark: mandatory
see.dbDriver = oracle.jdbc.driver.OracleDriver

#Database user identifier
#type: String
#remark: mandatory
see.dbUser = PMSEEADMIN

#Database user identifier
#type: String
#remark: mandatory
see.dbPwd = PMSEEADMIN

# end of file
```

tuning.properties

This file contains a list of configuration parameters and their current values.

```
#
=====
# Tuning parameter values
#
=====
#
# - formula can be used with previously defined parameters
# ex:
# jobProcessorEventSchedulerPeriod = 20
```

```

# jobProcessorEventSchedulerSSAMemorySize = 3 * 24 * (3600 /
jobProcessorEventSchedulerPeriod)

# -----
# Command parameters
# -----

#Internal check command period (in ms)
#0 means no internal check
#type: int
#remark: optional, default:90000
internalCheck.period = 90000

#Maximum retry period when a call made by a command has a busy response
# (in ms)
#type: long
#remark: optional, default:1000

maximum.retry.period.when.busy = 10000

# -----
# Visibroker parameters
# -----

visibroker.max.thread.pool.size = 40

# -----
# Input connector parameters
# -----

#Batch file monitor refresh period (in ms)
#type: long
#remark: optional, default:10000
batchFileConnector.fileMonitor.refreshPeriod = 10000

#Batch parameter monitor refresh period (in ms)
#type: long
#remark: optional, default:10000
batchFileConnector.parameterMonitor.refreshPeriod = 10000

#DDM subscription monitor check period (in ms)
#type: long
#remark: optional, default:60000
ddmConnector.subscriptionMonitor.checkPeriod = 60000
#DDM Connector Event Thread Pool Size
#type: int
#remark: optional, default:10
ddmConnector.event.threadPool = 10

#DDM Connector Event Queue Size
#type: int
#remark: optional, default:1000
ddmConnector.event.queue = 1000
# -----
# Command Lock Manager parameters
# -----

```

```
#Command Lock Manager activation status
#type: int
#values: 0: no lock mode, 1:Lock mono_instance, 2: RFU
#remark: optional, mode 1 is the default
commandLockManager.mode = 1

#Maximum number of managed locks. Once this limit is reached, new
incoming command will be rejected.
#By default this size is deduced from the maximum number of elements
allowed in the JobProcessor (times 3 because
#a subscription is generally locked upon the three identifiers iccid,
msisdn, imsi)
#WARNING: modifying this value will cause all the not terminated
commands to end with EXPIRED status
#Nevertheless these commands may be submitted again
#type: int
#remark: mandatory
commandLockManager.maxLocks = (see.spj.ssa.memorySize +
see.spj.ssa.swapSize)*3

#Maximum number of commands allowed to wait for a lock.
#WARNING: modifying this value will cause all the not terminated
commands to end with EXPIRED status
#Nevertheless these commands may be submitted again
#type: int
#remark: optional, default is 7 (min is 2, max is 15)
commandLockManager.maxLocksWaiters = 7

# -----
# Invocation Registry configuration
# -----

#IR activation status
#type: boolean
#remark: optional
see.ir.activation = true

#IR storage compression activation status
#type: boolean
#remark: optional, default true
see.ir.compression.activation = true

#Maximum number of rows before automatic commit: protection to avoid
rollback saturation.
#type: int
#remark: mandatory, default: 1000
see.ir.maxUpdatedRows = 1000

#Maximum number of rows strictly read for monitoring feature.
#type: int
#remark: mandatory, default: 1000
see.ir.maxReadRows = 500

#Delay between each replay
#type: int, Unit: second
```

```
#remark: mandatory, default: 5s
see.ir.delayBetweenReplay = 5

#Connection pool size dedicated to the monitoring (another connection
pool size is used for IR kernel)
#Only needed connections will be opened (depending on the application
load)
#type: int
#remark: mandatory, default: 25000
see.ir.monitoringConnPoolSize = 20

#Purge period used by automatic purge process
#All invocations TERMINATED since this period will be automatically
purged.
#type: int, Unit: day
#remark: mandatory, default: 7
see.ir.purgePeriod = 3

#The cursor timeout is the time limit for an idle cursor before it is
automatically
#closed to avoid resources saturation. (expressed in minute)
#type: int, Unit: minute
#remark: mandatory, default: 5
see.ir.cursorTimeout = 5

#Limit of opened cursors to restrict database resources usage
#type: int
#remark: mandatory, default: 100
see.ir.maxOpenedCursors = 200

#Database Connection pool size
#Only needed connections will be opened (depending on the application
load)
#type: int
#remark: mandatory, default: 10
see.ir.connPoolSize = 20

#Delay between each retry to restore database connection (s)
#type: int, Unit: second
#remark: mandatory, default: 30
see.ir.connRetryDelay = 10

#Timeout applied when attempt to realize a request over the Invocation
Registry.
#type: int, Unit: second
#remark: mandatory, default: 120
see.ir.requestTimeout = 120

#Flag to indicate if the invocation registry waits for database
dependencies before starting
#type: boolean
#remark: mandatory, default: false
see.ir.waitingDependencies = true

#Flag to indicate if the invocation history mode is enabled or not
#type: boolean
```

```
#remark: optional, default: true
see.ir.historicModeEnabled = true

# -----
# Invocation Manager parameters
# -----

# Queue size
# Type: int, Unit: none, Validity: [0;inf]
see.im.sequencerQueueSize = 1000

# Thresholds for proprietary memory flow control mechanism
# Once low threshold is reached, flow control is activated, so no more
# invocations are accepted (busy state). Once high threshold is
# reached, a
# garbage collector is made
# Type: float, Unit: none, Validity: [0;1]
# Remark: 1 means deactivated
see.im.lowMemoryHeapFlowControlThreshold = 0.9
see.im.highMemoryHeapFlowControlThreshold = 0.9

# Garbage collector call period
# Type: int, Unit: millisecond, Validity: [0;inf]
see.im.systemGcCallPeriod = 5000

# -----
# Job processor parameters
# -----

# --- Main ---

# Thread pool number of threads
# Type: int, Unit: none, Validity : [1;inf]
see.spj.threadPoolCapacity = 20

# Thread pool number of threads
# Type: int, Unit: none, Validity : [1;inf]
see.spj.broker.threadPoolCapacity = 5

# Default value for the shutdown timeout (in min).
# Type: int, Unit: min, Validity : [1;inf]
see.spj.shutdownTimeout = 10

# Job Queue size
# Type: int, Unit: none, Validity : [1;inf]
see.spj.JobQueueSize = 800

# --- Main SSA ---

#Flag to indicate if the SWAP tables are shared or not to truncate them
at start time
#type: boolean
#remark: mandatory
see.spj.ssa.areSwapTablesPrivate = true

#Database Connection pool size
```

```
#Only needed connections will be opened (depending on the application
load)
#type: int
#remark: mandatory
see.spj.ssa.connPoolSize = 20

#Delay between each retry to restore database connection (s)
#type: int, Unit: second
#remark: mandatory, default: 30
see.spj.ssa.connRetryDelay = see.ir.connRetryDelay

#Timeout applied when attempt to realize a request over the Invocation
Registry.
#type: int, Unit: second
#remark: mandatory, default: 120
see.spj.ssa.requestTimeout = see.ir.requestTimeout

#Flag to indicate if the invocation registry waits for database
dependencies before starting
#type: boolean
#remark: mandatory, default: false
see.spj.ssa.waitingDependencies = true

# Memory size
# The minimum size should not be less than
jobProcessorThreadPoolCapacity
# If less, jobProcessorThreadPoolCapacity is used instead of defined
value
# Type: int, Unit: none, Validity: [0;inf]
see.spj.ssa.memorySize = 40000

# Swap size
# Type: int, Unit: none, Validity: [0;inf]
see.spj.ssa.swapSize = 1000000

# Number of swap threads
# Type: int, Unit: none, Validity: [1;inf]
see.spj.ssa.swapThreadCount = 5

# Swap threshold
# Type: float, Unit: none, Validity: [0;1]
see.spj.ssa.swapThreshold = 0.9

# Swap step
# Type: int, Unit: none, Validity: [1;inf]
see.spj.ssa.swapStep = 20

# Event scheduler dedicated swap table name
see.spj.ssa.eventSchedulerSwapTableName = SWAP_EV_SCHED_LPM_AUTO

# Event dedicated swap table name
see.spj.ssa.eventSwapTableName = SWAP_EV_QUEUE_LPM_AUTO

# Job dedicated swap table name
see.spj.ssa.jobSwapTableName = SWAP_JOB_LPM_AUTO
```

```
# SWAP Type (MEMORY/DATABASE)
see.spj.ssa.swapType = DATABASE

# Enable/Disable memory key cache
see.spj.ssa.isKeyCacheEnable = true

# Default guard delay
# Guard delay for WAITING_EVENT_PSEUDO_STATE
# Type: int, Unit: second, Validity: [0;inf]
see.spj.ssa.defaultGuardDelay = 0

# Additional guard delay
# Type: int, Unit: second, Validity: [-1;inf]
# Remark: -1 means no additional delay and 0 means default value, which
# is 300 !!!
see.spj.ssa.additionalGuardDelay = -1

# FlowControlledSwappableStorageArea database reconnection retry delay
# (note
# that only job processor SSA is flow controlled)
# Type: int, Unit: second, Validity: [0;inf]
see.spj.fc.ssa.connRetryDelay = see.spj.ssa.connRetryDelay

# FlowControlledSwappableStorageArea SWAP thread count (note
# that only job processor SSA is flow controlled)
# Type: int, Validity: [0;inf]
see.spj.fc.ssa.swapThreadCount = 10

# FlowControlledSwappableStorageArea memory repartition ratio (note
# that only
# job processor SSA is flow controlled)
# Type: float, Unit: none, Validity: [0;1]
# Remark: Sum must be equal to 1
see.spj.fc.ssa.ratioEmergency = 0.1
see.spj.fc.ssa.ratioUrgent = 0.3
see.spj.fc.ssa.ratioNormal = 0.3
see.spj.fc.ssa.ratioLow = 0.3

# --- Event ssa ---

# Event queue memory size
# Type: int, Unit: none, Validity : [0;inf]
#
see.spj.evtq.memorySize = see.spj.ssa.memorySize*3

# Event queue swap size
# Type: int, Unit: none, Validity : [0;inf]
see.spj.evtq.swapSize = see.spj.ssa.swapSize*3

# --- Event scheduler ---

# Event scheduler SSA keys are timeslots: this is the check period for
# these
# timeslots
# Type: int, Unit: second, Validity: [1;inf]
see.spj.evtq.schedulerPeriod = 10
```



```
# --- Event scheduler SSA ---

# Memory size
# We consider 3 days-length timeslots in memory
# Type: int, Unit: none, Validity: [0;inf]
see.spj.evtq.ssa.memorySize = 3 * 24 * (3600 /
see.spj.evtq.schedulerPeriod)

# Swap size
# We consider 40 days-length timeslots in database
# Type: int, Unit: none, Validity: [0;inf]
see.spj.evtq.ssa.swapSize = 40 * 24 * (3600 /
see.spj.evtq.schedulerPeriod)

# Number of swap threads
# Type: int, Unit: none, Validity: [1;inf]
see.spj.evtq.ssa.swapThreads = 3

# end of file
```

utility_connectors.properties and utility_connectors.xml

The `utility_connectors.properties` and `utility_connectors.xml` files allow you to configure the service connectors used by Provisioning Manager to connect to other LinqUs Over-the-Air Suite products.

These files control the activation, deactivation or temporary pausing of the connection between Provisioning Manager and other LinqUs products.

utility_connectors.properties

The `utility_connectors.properties` file defines service connector names and their properties:

```
# =====
# utility_connector.properties : properties of the service connectors
# =====
# These properties are loaded in the files utility_properties.xml and
# commands.xml (which includes all commands statecharts)
# File format :
#  property_name=property_value
#
# Each '<property_value_to_set>' of the generic file must be replaced
# by the
# value reflecting your platform configuration (without '<', '>' or
# '"')
# Example :
#  pm_rca_name=RCA1
# =====
# General properties
# 'connector'_name = internal name of the service connector
# pm_"linqus_product"_name = name of the linqus product that is
# provisioned
# 'connector'_state = state of the service connector
# type = integer
# values = 1 for Active, 0 for Deactivated
# =====
```

```

# OTAManager Connector properties
# Example : otaManagerConnector/RCA1/1
# Caution: OTA Manager is mandatory, otaManagerConnector must be active
# =====
ota_connector_name=otaManagerConnector
pm_rca_name=RCA1
ota_connector_state=1

# SM Connector properties
# Example : smConnector/GCSM1/1
# =====
sm_connector_name=smConnector
pm_gcsm_name=GCSM1
sm_connector_state=1

# SCM Connector properties
# Example : scmConnector/SCM1/1
# =====
scm_connector_name=scmConnector
pm_scm_name=SCM1
scm_connector_state =1

# DM Connector properties
# Example : dmConnector/GCDM1/1
# =====
dm_connector_name=dmConnector
pm_gcdm_name=GCDM1
dm_connector_state=1

# PB Connector properties
# pm_pb_server_url : url of the PhoneBook JBoss server
# Example : pbConnector/10.10.128.12:8130/1
# =====
pb_connector_name=pbConnector
pm_pb_server_url=10.10.128.12:8130
pb_connector_state=1

```

utility_connectors.xml

```

<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN 2.0//EN" "http://
www.springframework.org/dtd/spring-beans-2.0.dtd">

<beans>

    <!-- Load custom s -->
    <import resource="custom/utility_connectors_CUSTOM.xml" />

    <!-- OTAManagerConnector -->
    <bean id="otaManagerConnector"
class="com.gemalto.pm.plugin.utilitycntr.ota.impl.OTAManagerConnectorI
mpl">
        <property name="name">
            <value>${ota_connector_name}</value>
        </property>

```

```

        <property name="productName" value="\${pm_rca_name}"/>
        <property name="state">
            <value>\${ota_connector_state}</value>
        </property>
    </bean>

    <!-- SMConnector -->
    <bean id="smConnector"
class="com.gemalto.pm.plugin.utilitycntr.sm.impl.SMConnectorImpl">
        <property name="name">
            <value>\${sm_connector_name}</value>
        </property>
        <property name="productName" value="\${pm_gcsn_name}"/>
        <property name="state">
            <value>\${sm_connector_state}</value>
        </property>
    </bean>

    <!-- SCMConnector -->
    <bean id="scmConnector"
class="com.gemalto.pm.plugin.utilitycntr.scm.impl.SCMConnectorImpl">
        <property name="name">
            <value>\${scm_connector_name}</value>
        </property>
        <property name="productName" value="\${pm_scm_name}"/>
        <property name="state">
            <value>\${scm_connector_state}</value>
        </property>
    </bean>

    <!-- DMConnector -->
    <bean id="dmConnector"
class="com.gemalto.pm.plugin.utilitycntr.dm.impl.DMConnectorImpl">
        <property name="name">
            <value>\${dm_connector_name}</value>
        </property>
        <property name="productName" value="\${pm_gcdm_name}"/>
        <property name="state">
            <value>\${dm_connector_state}</value>
        </property>
    </bean>

    <!-- PBConnector -->
    <bean id="pbConnector"
class="com.gemalto.pm.plugin.utilitycntr.pb.impl.PBConnectorImpl">
        <property name="name">
            <value>\${pb_connector_name}</value>
        </property>
        <property name="serverUrl" value="\${pm_pb_server_url}"/>
        <property name="state">
            <value>\${pb_connector_state}</value>
        </property>
    </bean>

    <!-- Bean instance of input connector loader -->

```

```

    <!-- To deploy a connector, you should register it towards this
         loader -->
    <!-- Be careful: only a loader declared as a singleton bean will
         trigger the loading of the input connectors -->
    <bean id="utilityConnectorLoader"
class="com.gemalto.pm.internal.api.utilitycntr.UtilityConnectorLoader"
init-method="deploy">
        <property name="registeredConnectors">
            <list>
                <ref bean="otaManagerConnector"/>
                <ref bean="smConnector"/>
                <ref bean="scmConnector"/>
                <ref bean="dmConnector"/>
                <ref bean="pbConnector"/>
            </list>
        </property>
    </bean>
</beans>

```

input_connectors.properties, input_connectors_states.ini, input_connectors.xml

The `input_connectors.properties`, `input_connectors_states.ini` and `input_connectors.xml` files allow you to configure the input connectors that are plugged into the Provisioning Manager.

input_connectors.properties

The `input_connectors.properties` file defines input connector names and their properties:

```

# =====
# input_connectors.properties : properties of the input connectors
# =====
# These properties are loaded in the file input_connectors.xml.
# File format :
#  property_name=property_value
#
# Each '<property_value_to_set>' of the generic file must be replaced
by the
# value reflecting your platform configuration (without '<', '>' or
'')
# Example :
#  pm_coreAPIConnectorName=coreAPIConnector1
#  pm_coreAPIConnector_pwd=gemalto40
# =====
# General properties
# pm_'connector'Name / pm_'connector'_pwd :
#   - used to connect to the linqUs products that are provisioned, and
to
#   monitor the connector's commands.
#   - connector name/password must match an existing framework user
#     (account identifier / account password)
#   - The connector name length must be < 20 characters (account name
limit)
# pm_'connector'VP :
#   - Validity Period of the commands issued by the connector

```

```

#      - type = integer
#      - value = expressed in minutes
# =====

# CoreAPI Connector properties
# Example : coreAPIConnector1/gemalto40
# =====
pm_coreAPIConnectorName=coreAPIConnector1
pm_coreAPIConnector_pwd=gemalto40

# Web services connector properties
# Example : wsConnector1/gemalto40
# =====
pm_webServicesConnectorName=wsConnector1
pm_webServicesConnector_pwd=gemalto40
pm_webServicesConnectorVP=15

# Interpreted Batch connector properties
# used for PM or TBL batch files
# Example : batchConnector1/gemalto40
# =====
pm_interpretedBatchConnectorName=batchConnector1
pm_interpretedBatchConnector_pwd=gemalto40
pm_interpretedBatchConnectorVP=1440

# TBL Direct Batch connector properties
# used for TBL direct batch (OTA Manager batch files format)
# Example : batchConnector2/gemalto40
# =====
pm_directBatchConnectorName=batchConnector2
pm_directBatchConnector_pwd=gemalto40
pm_directBatchConnectorVP=1440

# DDM connector properties
# pm_ddmCommand (type=String) Command triggered by DDM connector on
# IMEI Event reception.
# It is also possible to trigger several commands in parallel, separating
# these commands
# by a comma character as shown in Example 2.
# Example 1 : ddmConnector1/gemalto40/GCDT1/4320/
ManageDDMChangeImeiEvent
# Example 2 : ddmConnector1/gemalto40/GCDT1/4320/Command1, Command2
#
# pm_ddmRecovery (type=boolean)
# Enable (true) or Disable (false) the recovery mechanism
# from DDM input connector
#
# pm_ddmInfoEventProcessing (type=boolean)
# If set to true, all events received from Device Tracking will be
# processed,
# including IMEI Info events.
# If set to false, only IMEI Change events and Update Card Data Info
# events
# received from Device Tracking will be processed.
# =====
pm_ddmConnectorName=<pm_ddmConnectorName>

```

```

pm_ddmConnector_pwd=<pm_ddmConnector_pwd>
pm_ddm_name=<pm_ddm_name>
pm_ddmConnectorVP=4320
pm_ddmCommand=ManageDDMChangeImeiEvent
pm_ddmRecovery=true
pm_ddmInfoEventProcessing=true

```

input_connectors_states.ini

The `input_connectors_states.ini` file controls the activation, deactivation or temporary pausing of an input connector.

This file is read when the Provisioning Manager first starts and is updated each time an input connector is deactivated or activated through the GUI.

```

#Input connector states:
#key is like: inputcntr.state.<connector_name>=<state>
#state can be 0 = deactivated and 1 = activated
#This file is updated each time any input connector state is changed.
#Thu Feb 28 15:45:21 CET 2008
inputcntr.batchConnector1.state=1
inputcntr.ddmConnector1.state=0
inputcntr.batchConnector2.state=0
inputcntr.wsConnector1.state=1
inputcntr.coreAPIConnector1.state=1

```

input_connectors.xml

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN 2.0//EN"
    "http://www.springframework.org/dtd/spring-beans-2.0.dtd">
<beans>
    <!-- Load custom input connectors -->
    <import resource="custom/input_connectors_CUSTOM.xml" />

    <!-- CoreAPIConnector -->
    <!-- This connector is mandatory for the CoreAPI implementation of
        the Provisioning Manager -->
    <bean id="coreAPIConnector1"
class="com.gemalto.pm.plugin.inputcntr.coreapi.CoreAPIConnector">
        <property name="name" value="{pm_coreAPIConnectorName}"/>
        <property name="userPassword"
            value="{pm_coreAPIConnector_pwd}"/>
    </bean>

    <!-- WebServicesConnector -->
    <bean id="webServicesConnector1"
class="com.gemalto.pm.plugin.inputcntr.ws.WebServicesInputConnector">
        <property name="name" value="{pm_webServicesConnectorName}"/>
        <property name="userPassword"
            value="{pm_webServicesConnector_pwd}"/>
        <property name="validityPeriod"
            value="{pm_webServicesConnectorVP}"/>
    </bean>

    <!-- Interpreted BatchFileConnector -->
    <bean id="batchConnector1"
class="com.gemalto.pm.plugin.inputcntr.batchfile.BatchFileConnectorImp
1">

```

```

        <property name="name"
            value="\${pm_interpretedBatchConnectorName}"/>
        <!-- Connector file mode. Available modes are: -->
        <!-- Interpreted: Interpreted files connector (managing standard
PM and TBL formats) -->
        <!-- Direct: Direct files connector (managing OTA card and
security formats) -->
        <property name="fileMode" value="Interpreted"/>
        <!-- Indicates if commands sent with this connector are
monitorable -->
        <property name="monitorableCommands" value="true"/>
        <property name="userPassword"
            value="\${pm_interpretedBatchConnector_pwd}"/>
        <property name="validityPeriod"
            value="\${pm_interpretedBatchConnectorVP}"/>
    </bean>

    <!-- Direct BatchFileConnector -->
    <bean id="batchConnector2"
class="com.gemalto.pm.plugin.inputcntr.batchfile.BatchFileConnectorImp
1">
        <property name="name" value="\${pm_directBatchConnectorName}"/>
        <!-- Connector file mode. Available modes are: -->
        <!-- Interpreted: Interpreted files connector (managing standard
PM and TBL formats) -->
        <!-- Direct: Direct files connector (managing OTA card and
security formats) -->
        <property name="fileMode" value="Direct"/>
        <!-- Indicates if commands sent with this connector are
monitorable -->
        <property name="monitorableCommands" value="true"/>
        <property name="userPassword"
            value="\${pm_directBatchConnector_pwd}"/>
        <property name="validityPeriod"
            value="\${pm_directBatchConnectorVP}"/>
    </bean>

    <!-- DDMConnector -->
    <bean id="ddmConnector1"
        class="com.gemalto.pm.plugin.inputcntr.ddm.DDMConnector">
        <property name="name" value="\${pm_ddmConnectorName}"/>
        <property name="userPassword" value="\${pm_ddmConnector_pwd}"/>
        <property name="productName" value="\${pm_ddm_name}"/>
        <property name="validityPeriod" value="\${pm_ddmConnectorVP}"/>
        <property name="issuedCommandName" value="\${pm_ddmCommand}"/>
    </bean>

    <!-- Bean instance of input connector loader -->
    <!-- To deploy a connector, you should register it toward this loader
-->
    <!-- Be careful: only a loader declared as a singleton bean will
trigger the loading of the input connectors -->
    <bean id="inputConnectorLoader"
class="com.gemalto.pm.internal.api.inputcntr.InputConnectorLoader"
        init-method="deploy">
        <property name="registeredConnectors">

```

```
        <list>
            <ref bean="coreAPIConnector1"/>
            <ref bean="webServicesConnector1"/>
            <ref bean="batchConnector1"/>
            <ref bean="batchConnector2"/>
            <ref bean="ddmConnector1"/>
        </list>
    </property>
</bean>
</beans>
```

commands.xml and commands.properties

The `commands.properties` and `commands.xml` files allow you to configure the provisioning commands that are plugged into the Provisioning Manager.

commands.properties

This file defines commands' properties. These properties are loaded in the file `commands.xml`.

```
# =====
# commands.properties : properties of the commands
# =====
# These properties are loaded in the file commands.xml (which includes
all
# commands statecharts)
# =====
# File format :
#   property_name=property_value
# =====

# execScript property
# targetType = type of the target used for the destination of OTAManager
# execScript service.
#   type = string
#   value = msisdn, iccid or imsi
#       default = iccid
# =====
cmd.execScript.targetType=iccid
```

commands.xml

This file is the main entry point for commands' state charts and contains Provisioning Manager command definitions. It includes state charts for all standard and custom provisioning commands.

Error Messages and Exceptions

The Provisioning Manager module generates both general and Provisioning Manager-specific error messages, as defined in the following tables. These error messages are copied into the Base System database when the product is installed.

General Error Messages

Table 20 - General Error Messages

Error code	Exception name	Description
PM-1	UnMappedException	Unhandled error
PM-10	UnknownCommandException	Unknown provisioning command
PM-11	CommandHandlingException	Error occurred during command execution
PM-12	MissingParameterException	Mandatory command parameter is missing
PM-13	InvalidCommandParameterException	Command parameter is invalid
PM-14	InconsistentCommandParameterException	Inconsistent subscription identification. Products are not provisioned with the same values
PM-15	InconsistentSubscriptionInProductsException	Inconsistent subscription between products
PM-100	SubscriptionDoesNotExistException	Subscription does not exist on product
PM-101	SubscriptionAlreadyExistException	Subscription already exists on product
PM-102	AlreadyAllocatedMsisdnException	MSISDN already allocated to another card
PM-103	AlreadyAllocatedImsiException	IMSI already allocated to another card
PM-1000	XMLParsingException	General XML parsing exception
PM-1010	TBLException	General TBL (Tool for Batch Load) exception. Command cannot be replayed. See exception message and error code of embedded exception for more details
PM-10000	InvalidProvisionedProductConfigException	Invalid provisioned product configuration. A trigger may not be activated
PM-10001	UtilityConnectorDeactivatedException	Connection to a targeted product is deactivated
PM-10002	UtilityConnectorDisconnectedException	Connection to a targeted product is down

Table 20 - General Error Messages (continued)

Error code	Exception name	Description
PM-10003	UtilityConnectorProductAccessException	Access to the targeted product has failed
PM-10004	UtilityConnectorProductInvocationException	Unable to request service execution on a product
PM-10005	FileTransferException	File transfer to product has failed
PM-10006	MaxLockLimitReachedException	Unable to accept the new invocation as the maximum number of locks has been reached
PM-10007	LockRequestException	Unable to set the locks requested by the command
PM-10008	TemporaryCommandHandlingException	Unable to accept the new command execution due to a temporary internal error. Server is probably busy
PM-10009	ProductExecutionException	A product has returned an error, caused by an external resource usage, while executing an operation requested by a provisioning command
PM-10200	SubscriptionNotDeletedException	Subscription has not been deleted on product
PM-10201	SubscriptionNotInactivatedException	Subscription has not been inactivated on product
PM-10202	SubscriptionNotUpdatedException	Subscription has not been updated on product
PM-10203	IllegalSubscriptionException	Subscription state found on product is incompatible with requested command operation
PM-10204	SubscriptionNotActivatedException	Subscription has not been activated on product
PM-10210	LockedSubscriptionException	Command has tried to execute an action on a card locked by a product. The server has refused the requested action
PM-10300	OTACardNotDeletedException	SIM card has not been deleted on OTA Manager server
PM-10301	OTACardNotInactivatedException	SIM card has not been inactivated on OTA Manager server
PM-10302	OTACardNotUpdatedException	SIM card has not been updated on OTA Manager server
PM-10303	OTACardNotActivatedException	SIM card has not been activated on OTA Manager server
PM-11000	ConnectionException	(Web service client) Report that the Web service application is not granted to access server. Check that the user used by the Web service application has the necessary access rights to the server
PM-11001	ConnectionException	(Web service client) Report that the Web service application cannot access the required connector on the server side
PM-11002	ConnectionException	(Web service client) Report that the Web service end user has provided an invalid session token
PM-12000	AccessDeniedException	(Core API client) Report that the connected client does not have the necessary access rights to perform the operation
PM-12001	CommandMonitoringException	(Core API client) Command manager has encountered a problem while performing the operation

Configuring the Web Services Component

LinqUs Provisioning Manager uses a Tomcat 5.0.28 server for the Web services component

See the *LinqUs Provisioning Manager 4.1 Installation Guide* for details.

Note: A separate Tomcat 3.3.2 server is used for the LinqUs Over-The-Air Suite platform GUI.

pm.properties

This file contains parameters used by the Web services component to call the Provisioning Manager platform, for example:

- CORBA parameters
- Identifier of the Provisioning Manager product to use
- Name of the Web service input connector
- Default user account name and password.

By default (installer's configuration), this file is located in the folder `<jakarta-tomcat-5.0.28_HOME>/webapps/<LPM_instance_name>/WEB-INF/classes`.

Default pm.properties file:

```
#####
# Visibroker #
#####
### Visibroker port to connect to LinqUs platform (ex: 16047).
vbroker.agent.port=17085
### Visibroker parameters necessary to tomcat JVM (ex: /opt/vbroker/
var).
borland.enterprise.licenseDir=/opt/vbroker/var
### Visibroker parameters necessary to tomcat JVM (ex: /opt/vbroker/
license).
borland.enterprise.licenseDefaultDir=/opt/vbroker/license

container.package=CORBA

#####
# Product #
#####
### Name of LinqUs Provisioning Manager product to use (ex: PM).
```

```
pm.product.name=PM
### Name of the Web service input connector (ex: wsConnector1).
pm.webServices.name=wsConnector1
### User name to connect to LinqUs platform (ex: admin).
pm.connection.username=administrator
### User password to connect to LinqUs platform (ex: admin).
pm.connection.password=administrator_pwd
```

The prefix of the `pm.properties` file (“pm”) is specified in each Web service call. This allows you to create multiple Web service component configurations. For example, each instance of the Provisioning Manager product (PM1, PM2, and so on) can be targeted by a different file (`PM1.properties`, `PM2.properties` and so on).

Example: `pm1.properties` (PRODUCT part)

```
#####
# Product #
#####
### Name of LinqUs Provisioning Manager product to use (ex: PM).
pm.product.name=PM1
### Name of the Web service input connector (ex: wsConnector1).
pm.webServices.name=wsPM1
### User name to connect to LinqUs platform (ex: admin).
pm.connection.username=administrator
### User password to connect to LinqUs platform (ex: admin).
pm.connection.password=administrator_pwd
```

Note: TBL provisioning commands use the file name `pm.properties`. This file name cannot be configured.

A

agent
 ORB **25, 26**
 SNMP **23**
 audit trail
 file format **19**

B

Base System database **51**
 batch file processing **6, 11**
 batchload input connector **13**
 billing ticket file format **20**

C

CAT-TP
 as default OTA protocol **34**
 CCI interface **5**
 command lock manager **15**
 CORBA **3**
 Core API **5**
 counters, SNMP **26**

D

database
 Base System **51**
 connection **25**
 indexes, rebuilding **6**
 invocation identifier key field **17**
 keys **v**
 management **6**
 OTA Manager **12**
 status **27**
 deactivating service connectors **5**
 deleting lock file **3**
 Device Manager
 as provisioning source **vi**

E

error messages
 general **51**
 general errors **51**
 exception
 "out of memory" **14**
 name **24**
 exceptions **51**

Exec script
 default send mode **34**

F

format
 audit trail **19**
 billing ticket **20**
 log file **19**
 Framework **1**

G

`gemconnect.cfg` script **1**
 general error messages **51**

I

IN_PROCESS state **4**
 input connector
 batchload **13**
 stopping **4**
 user account for **4**
 Web service **53**
 Input connectors
 viewing status of **14**
 input connectors
 associated user account **4**
 configuring **46**
 monitoring **17**
 stopping **4**
 stopping input flow **4**
 time frame **34**
 invocation identifier **17**

J

Java Virtual Machine (JVM) **1**

L

`lock.db` file **14**
 locks
 command manager **15**
 file **14**
 file, deleting **3**
 managing **14**
 storing **15**
 tuning parameters for **15**
 log file **17, 19**

log file format **19**
logging **19**

M

Management Information Base (MIB)
 See *MIB*
MIB structure **23**

O

ORB agent port **25, 26**
OSAGENT **2, 3**
OSAGENT script **2**
OTA Manager **12**
OTA protocol, default **34**

P

persistent traps
 platform **26**
 SNMP **26**
Phonebook Backup **32**
platform
 persistent traps **26**
 status (SNMP traps) **26**
product
 resetting **3**
 restarting **14**
 starting and stopping a **1**
product parameters
 changing values of **33**
 default values **34**
 THROUGHPUT **4, 34**
provisioning
 command locks file **3**
 recommendations **5**
 stopping **4**

R

recommendations for provisioning **5**
resetting the Provisioning Manager **3**

S

scripts
 gemconnect.cfg **1**
 OSAGENT **2**
 TOMCAT **2**

see.spj.threadPoolCapacity parameter **5**
service connectors
 configuring **43**
 deactivating **5**
shutdown.sh script **2**
SNMP
 agent **23**
 counters **26**
 persistent traps **26**
 traps **23**
starting
 products **1**
startup.sh script **2**
stopping
 databases **3**
 input connectors **4**
 provisioning **4**
storing locks **15**
subscriptions
 creating **5**
 updating **10**

T

THROUGHPUT product parameter **4, 34**
time frame for running input connector **34**
TOMCAT script **2**
Tomcat Web servers, use of two **2**
tuning locks **15**
tuning.properties file **5**

U

user accounts
 creating **3**
 of input connectors **4**
user profile, adding Provisioning rights to **3**
utility_connectors.properties file **5**

V

validity period, default **34**
Visibroker thread pool **5**

W

Web services component **2**

