ConnecTerra®

RFTagAware™ Enterprise Server 1.0 Administrator's Guide

December 2005

Contact Information

BEA Systems, Inc.
4 Van de Graaff Drive
Burlington, MA 01803
Phone: 781-993-7100
Facsimile:781-993-7601
Web Site: www.bea.com

Preface

Purpose of This Manual

This manual describes how to install, configure, and use the RFTagAware™ Enterprise Server.

Audience

- System administrators and other IT professionals responsible for deploying Enterprise Server components.
- Software engineers and other developers using the APIs provided with the Enterprise Server.
- IT professionals maintaining enterprise sites running the Enterprise Server.

Users should have knowledge of systems, networking, and how to install applications under Microsoft® Windows or UNIX® systems.

Related Documents

- *Creating WebLogic Domains Using the Configuration Wizard*
- *Deploying Applications to WebLogic Server*
- *Configuring and Managing JMS*
- *RFTagAware Reader Configuration Guide*
- *RFTagAware Programmer Guide*

What's in This Manual

- Chapter 1: Introduction

  This chapter provides an overview of the Enterprise Server.

- Chapter 2: Installing and Configuring Enterprise Server

  This chapter describes how to install, configure components within, start, and stop the Enterprise Server.

- Chapter 3: Using EPCIS

  This chapter describes how to use the EPC Information Service, an Enterprise Server component.

- Chapter 4: Using the Provisioning Service

  This chapter describes how to use the Provisioning Service, an Enterprise Server component.

- Chapter 5: Data Reference

  This chapter describes the database schemas and views exposed by Enterprise Server components.

- Chapter 6: Glossary

  This chapter describes common terms used by the Enterprise Server.

Contacting Technical Support

For technical support Monday-Friday 9:00am-5:30pm Eastern Time, call 781-993-7900, or send email to support@connecterra.com.

BEA Systems, Inc.
4 Van de Graaff Drive
Burlington, MA 01803

Phone: +1-781-993-7100
Facsimile: +1-781-993-7601

Contents

Chapter 1: Introduction

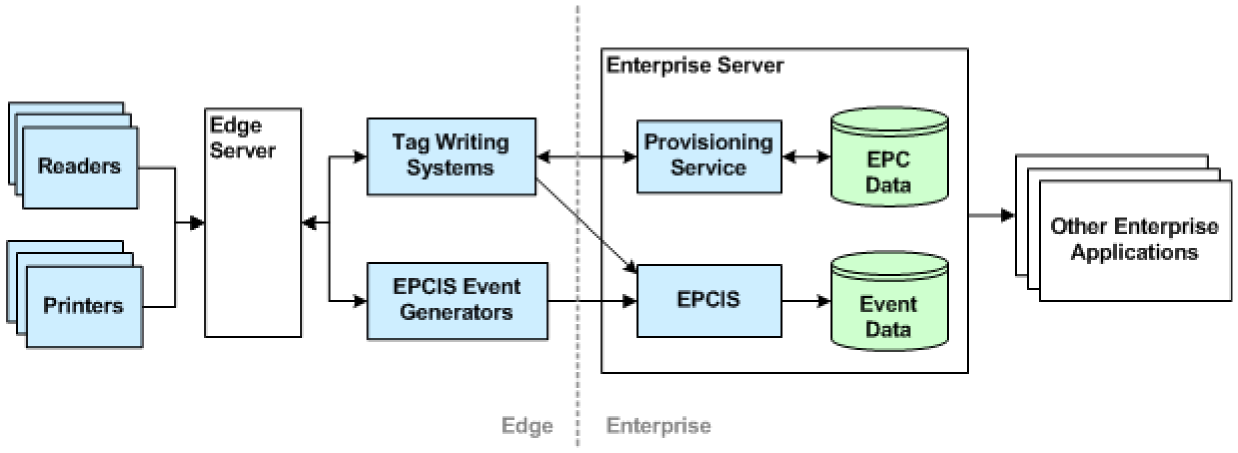
Contents

This chapter provides an overview of the Enterprise Server.

Enterprise Server Overview

The RFTagAware™ Enterprise Server is BEA's solution for providing an enterprise-level view of RFID-labeled objects moving through the supply chain. RFID labels uniquely identify objects by means of an Electronic Product Code (EPC) written to the tag. Tracking tagged items as they move through the supply chain from manufacturer to shipper to warehouse to retail store generates large quantities of data at the edge. The enterprise-relevant data is then extracted and sent on to the Enterprise Server, which stores it in a form that is useful to other enterprise applications and provides interfaces to access the data.

Enterprise Server Components



The Enterprise Server 1.0 has two main components:

- The EPC Information Service (EPCIS) receives notifications of EPCIS events (observations, aggregations, quantity changes, and association of EPCs with business transactions) and makes that data available to other enterprise applications. This component allows end-to-end tracking of EPC-bearing objects (for example, cases or pallets of your product) from the first observation after the RFID label was applied until the product is displayed for sale.
- The Provisioning Service provides centralized management of EPC provisioning for one or more tag writing systems. By maintaining pools of EPCs in a central location, the likelihood of duplicate or misallocated EPC blocks is reduced. EPC allocation for all your tag writing systems is accomplished by connecting to the Provisioning Service, and data on the rate of allocation to various systems is immediately available.

Standards Compliance

The Enterprise Server is designed to be compliant with all relevant standards of EPCglobal (formerly known as the Auto-ID Center). We are committed to revising our software offerings so that they are always in compliance with these standards.

The Enterprise Server supports the *EPCglobal Tag Data Standards*. This document, available on the EPCGlobal Web site at *http://www.epcglobalinc.org*, governs the bit-level encoding of object identity and other information onto RFID tags. It also specifies a URI-based syntax for exchange of tag data between software application components, and a second URI-based syntax for the description of filtering patterns. The Enterprise Server fully implements all of these standards.

In addition, the Enterprise Server supports the current draft of EPCGlobal's *EPCIS Specification*, which defines Version 1.0 of EPC Information Services (EPCIS). This specification provides a framework for sharing of EPC data both within and across enterprises, to obtain an enterprise-level view of the disposition of EPC-bearing objects. The enterprise-level view associates a relevant business context with the pure EPC data collected by observations along the supply chain.

What's Next

The remaining chapters in this document describe how to install, configure, and use the Enterprise Server, and how to write programs that interact with its components.

Chapter 2: Installing and Configuring Enterprise Server

Contents

This chapter describes how to install and configure the Enterprise Server

System Requirements

The main components of the Enterprise Server system are:

- EPCIS (EPC Information Service) – Used to track RFID-tagged objects and associate business-level information with them. EPCIS provides capture and query APIs, and database tables for data persistence.
- Provisioning Service – Used to coordinate automatically allocating blocks of EPCs to tag writing systems. The Provisioning Service provides a Web interface to create selection rules and monitor pools, as well as an API to check in and check out EPCs, and database tables for data persistence.

The Enterprise Server Installer places all the components on a single machine, sets configuration parameters and provides documentation. The Enterprise Server can be configured manually to run in a distributed configuration (that is, where all components are not installed on the same machine); please consult your authorized account representative for assistance if you require this option.

The Enterprise Server is supported on the following platforms:

- Microsoft Windows 2000 Professional, Windows 2000 Server, Windows XP, and Windows 2003 Server.
- Intel Pentium-compatible running Red Hat Linux (Version 8, Version 9, or Fedora Core 3).

Prerequisite Software

You will need to install BEA WebLogic Server 9.1 before installing Enterprise Server: This installation will provide access to the WebLogic application server and PointBase database. For production systems, you will need to use a database suited to your organization's data.

In addition, there must be a Web browser and a network connection present on the system where the Enterprise Server is installed, and on any system where the Provisioning Service Administrative Console will be run. The Provisioning Service Administrative Console is a standalone Web client used to configure and monitor Provisioning Service activity.

If you are installing on Linux, you must use a graphical environment to run the installer. Specifically, you must have a valid X Window System `DISPLAY` variable set in your environment. If you are installing onto a Fedora Core 3 machine, you must add the `xorg-x11-deprecated-libs` package before starting the installer.

Installing Enterprise Server

This section shows you how to install the Enterprise Server.

- Directory Structure Concepts (page 2-3)
- Running the Installer (page 2-4)

Directory Structure Concepts

Defaults and Allowed Install Locations

The Enterprise Server's default installation directory is:

- Windows: `C:\Program Files\ConnecTerra\EnterpriseServer\<version>`
- Linux: `/opt/ConnecTerra/EnterpriseServer/<version>`

This directory contains the files for all the components of the version of Enterprise Server you will be installing, and supporting files.

You can choose a different primary install location for the Enterprise Server. Please contact your account representative for questions regarding site-specific installation requirements.

Directory Structure

The installer creates four main subdirectories beneath the Enterprise Server install directory:

- `common`

  Location for scripts and libraries that are common to all Enterprise Server components.
- `deploy`

  Location for Enterprise Server modules that are deployed within WebLogic Server. This location also includes property and configuration files.
- `EPCIS`

  Location for scripts, executables, documentation, and libraries specific to the EPCIS component of the Enterprise Server.
- `ProvisioningService`

  Location for scripts, executables, documentation, and libraries specific to the Provisioning Service component of the Enterprise Server.

Shortened Pathnames in Documentation

The Enterprise Server documentation provides shortened pathnames to refer to specific locations within the Enterprise Server and WebLogic Server installation directories, as follows:

- `ENTERPRISE_HOME = C:\Program Files\ConnecTerra\EnterpriseServer\`*`<version>`*
- `EPCIS_HOME = C:\Program Files\ConnecTerra\EnterpriseServer\`*`<version>`*`\EPCIS`
- `EPCPS_HOME =`
  `C:\Program Files\ConnecTerra\EnterpriseServer\`*`<version>`*`\ProvisioningService`
- `WL_HOME = C:\bea\weblogic91`
- `DOMAIN_HOME = C:\bea\user_projects\domains\`*`<domain>`*
- `COMPLIANCE_HOME = C:\Program Files\ConnecTerra\Compliance\`*`<version>`*
- `RFTA_HOME = C:\Program Files\ConnecTerra\RFTagAware\`*`<version>`*

Note: These shortcuts provide easy-to-remember directory references throughout the document, and should not be confused with WebLogic Server environment variables.

Running the Installer

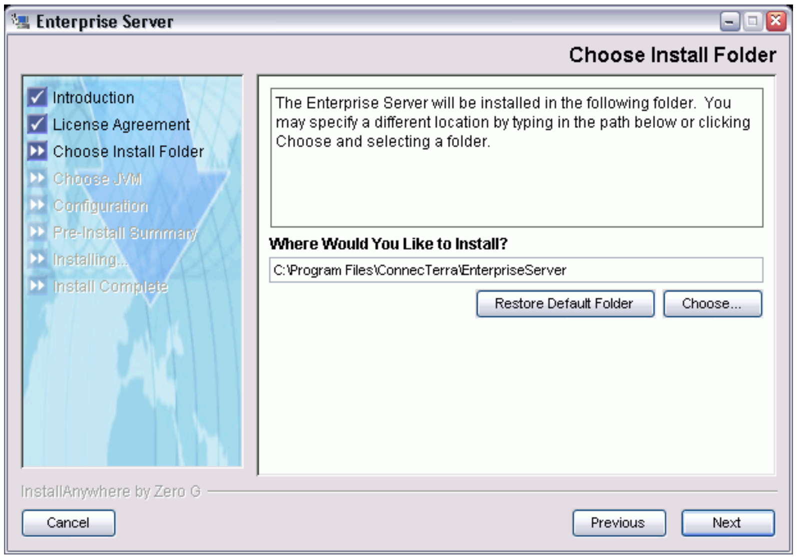
This section provides screenshots and descriptions of a sample installation. Your choices when running the installer may differ from the sample shown here.

1. Insert the Enterprise Server installation CD into the machine where you want to install (or download the installer), and run the install program for your platform:

   (Windows) `win_esinstall_1.0.exe`
   (Linux) `lin_esinstall_1.0.bin`

2. The installer guides you through the introduction screen and license agreement.

   The Choose Install Folder screen displays.



The Enterprise Server will be installed in the following folder. You may specify a different location by typing in the path below or clicking Choose and selecting a folder.

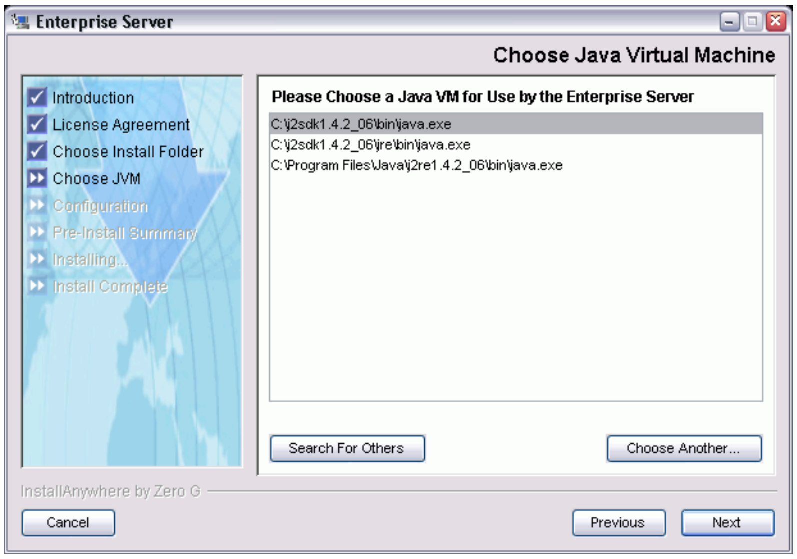
Where Would You Like to Install?

C:\Program Files\ConnecTerra\EnterpriseServer

3. If you do not want to install to the default directory, click the **Choose** button to browse to the directory where you want to install the software. Select the directory from the dialog that displays and click **OK**. (You can also type a path and file name to override the default location provided.)

   Click **Next** to proceed with the installation.

4. The Choose Java Virtual Machine screen displays.

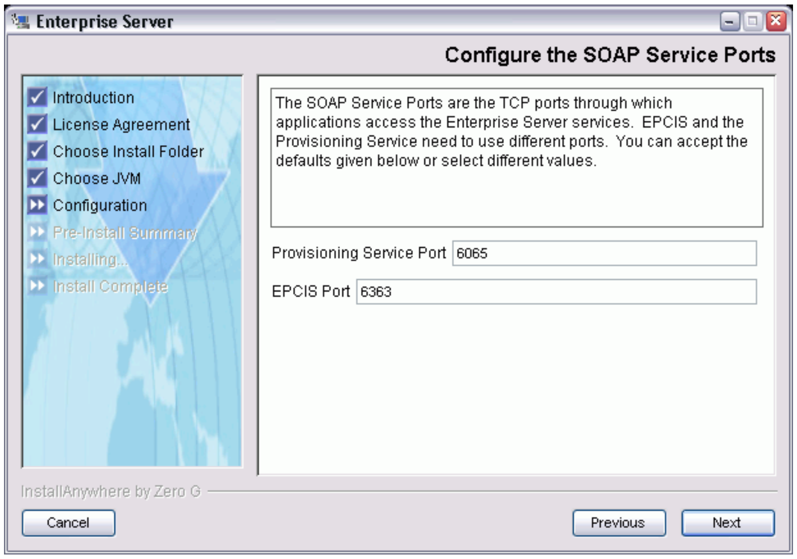


Please Choose a Java VM for Use by the Enterprise Server

C:\j2sdk1.4.2_06\bin\java.exe
C:\j2sdk1.4.2_06\jre\bin\java.exe
C:\Program Files\Java\j2re1.4.2_06\bin\java.exe

   Choose an existing Java Virtual Machine from the list, or use the buttons to find another Java Virtual Machine to use. When you have finished, click **Next**.

5. The Configure SOAP Service Ports screen displays.



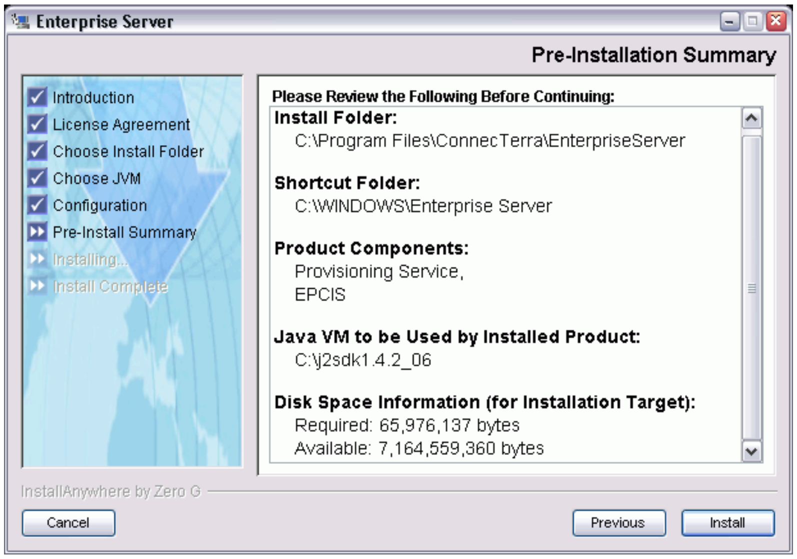
The SOAP Service Ports are the TCP ports through which applications access the Enterprise Server services. EPCIS and the Provisioning Service need to use different ports. You can accept the defaults given below or select different values.

Provisioning Service Port 6065

EPCIS Port 6363

   Enter the TCP port numbers that applications will use to access EPCIS and the Provisioning Service, or accept the default port numbers, then click **Next**.

6. The Pre-Installation Summary screen displays.



Please Review the Following Before Continuing:

Install Folder:
C:\Program Files\ConnecTerra\EnterpriseServer

Shortcut Folder:
C:\WINDOWS\Enterprise Server

Product Components:
Provisioning Service,
EPCIS

Java VM to be Used by Installed Product:
C:\j2sdk1.4.2_06

Disk Space Information (for Installation Target):
Required: 65,976,137 bytes
Available: 7,164,559,360 bytes

Click **Install** to proceed with the installation

7. Click **Done** when the Install Complete screen displays.

Configuring Enterprise Server

This section provides configuration information for a sample installation, as shown above. If you have questions about more complex configurations, please consult your account representative for assistance.

There are several initial configuration steps you will need to perform before starting the Enterprise Server system.

1. Create and initialize the Enterprise Server database.

   See Configure the Enterprise Server Database on page 2-6.

2. Perform WebLogic Server setup tasks.

   See Configure WebLogic Server on page 2-8.

Configure the Enterprise Server Database

To create and initialize the database used by the Enteprise Server using PointBase:

1. Start the PointBase Server by running the `WL_HOME\common\eval\pointbase\tools\startPointBase` script.

2. Start the PointBase Console by running the `WL_HOME\common\eval\pointbase\tools\startPointBaseConsole` script.

3. Create the `ent_server` database and `es` user using the PointBase Console:
   a. Create a database named `ent_server` using the URL:
      `jdbc:pointbase:server://localhost:9092/ent_server`
   b. Log in as user `PBPUBLIC`, with password `PBPUBLIC`.
   c. Create user `es` with password `es` (using **DBA > Create > User...**).
   d. Disconnect from the database (using **DBA > Disconnect from Database**).
   e. Reconnect to the database as user `es` (using **DBA > Connect to Database**).
   f. Create a schema called `es` (using **DBA > Create > Schema...**).
4. Run the `ENTERPRISE_HOME\common\bin\init-es-db-tables` script to create the database schema and tables.
5. Shut down the PointBase Console.

To create and initialize the database used by the Enteprise Server using other database tools:

1. Obtain the JDBC driver for the database tool you are using, and copy it into `ENTERPRISE_HOME\common\lib`. Note that Oracle and Sybase JDBC drivers are included with the WebLogic Server installation, and can be found in `WL_HOME\server\lib`.

   For information on configuring using third-party JDBC drivers with WebLogic Server, see *http://e-docs.bea.com/wls/docs91/jdbc_admin/third_party_drivers.html.*
2. Modify the database creation scripts in `EPCIS_HOME\bin` and `EPCPS_HOME\bin` to substitute the correct JDBC driver for `%JDBC_LIB%\pbclient51.jar`.
3. Create a database and user for use with Enterprise Server.

   Note: Even if your database tool has a default user such as `SYSTEM` or `PUBLIC`, we strongly recommend that you create a new user for use with the Enterprise Server.
4. Edit `ENTERPRISE_HOME\common\lib\kodo.properties`, commenting out the active database properties and uncommenting the properties for the database tool you are using. Verify that the property values for `ConnectionURL`, `ConnectionUserName`, and `ConnectionPassword` are correct for your system.
5. Make the same changes to the values (`<config-property-value>`) of `ConnectionURL`, `ConnectionUserName`, and `ConnectionPassword` in the following two files:
   ```
   ENTERPRISE_HOME\deploy\epcis.ear\kodo.rar\META-INF\ra.xml
   ENTERPRISE_HOME\deploy\epc-provisioning.ear\kodo.rar\META-INF\ra.xml
   ```
6. If your database supports the select for update query with order clause, edit the two `ra.xml` files to remove the property value for `kodo.jdbc.sql.DBDictionary`. This property value is set to `SupportsLockingWithOrderClause=false` for databases that do not support this feature.
7. Run the `ENTERPRISE_HOME\common\bin\init-es-db-tables` script to create the database schema and tables.

Configure WebLogic Server

Use the following steps to configure WebLogic Server 9.1 to run Enterprise Server 1.0 components.

1. Use the Configuration Wizard (accessed via the `WL_HOME\common\bin\config` script) to create a new WebLogic Server domain called `esdomain`. Choose *Production Mode* for the Domain Startup Mode, and *JRockit SDK 1.5.0_04* for the JDK Selection.

   Note: If you choose a user name other than `weblogic`, you will need to edit the Enterprise Server configuration file to change the default name there. Edit `weblogic.xml` in `ENTERPRISE_HOME\deploy\provisioning.ear\epcps-webclient.war\WEB-INF` to substitute the user name you chose:

   ```
   <principal-name>weblogic</principal-name>
   ```

   For more information, refer to the domain creation instructions provided with the WebLogic Server 9.1 documentation at *http://e-docs.bea.com/common/docs91/confgwiz/index.html.*

2. Start WebLogic Server for the new domain by running the `DOMAIN_HOME\bin\startWebLogic` script.

3. Once the server has started, use a Web browser to navigate to the WebLogic Administration Console (usually accessible at *http://localhost:7001/console*) and log in using the user name and password you created in step 1.

4. Deploy the Enterprise Server components in WebLogic Server:

   a. Navigate to Deployments under your domain in the Domain Structure tree.

   b. In the Change Center of the Administration Console, click **Lock & Edit**.

   c. Click **Install**.

   d. Navigate to `ENTERPRISE_HOME\deploy`.

   e. Install `epcis.ear` as an application (that is, select **Install this deployment as an application** and use the other defaults provided).

   f. Install `epc-provisioning.ear` as an application (that is, select **Install this deployment as an application** and use the other defaults provided).

   g. In the Change Center, click **Activate Changes**, and wait for the changes to be activated.

5. Create a JMS Server, Module, and targets for Enterprise Server components:

   a. In the Change Center of the Administration Console, click **Lock & Edit**.

   b. Create a JMS Server, with server target `AdminServer`.

   c. Create a JMS Module.

   d. Create a new resource (a Topic), with JNDI name `EPCISTopic`.

   e. Target the topic to the JMS Server name. This topic will be used by the deployed EPCIS application for communication via JMS.

f. Create a new resource (a Queue), with queue name `epcpsBrokerMessages`.

g. Target the queue to the JMS Server name. This queue will be used by the deployed Provisioning Service application for communication via JMS.

h. In the Change Center, click **Activate Changes**.

For more information, refer to the JMS configuration instructions provided with the WebLogic Server 9.1 documentation at *http://e-docs.bea.com/wls/docs91/jms_admin/basic_config.html*. The links to *JMS Server Configuration* and *JMS System Module Configuration* will provide detailed configuration instructions.

6. Start the Enterprise Server component deployments so that they service all requests. When their state changes to **Active** in the Summary of Deployments on the WebLogic Administration Console, the component is deployed.

Starting Enterprise Server

The instructions given below assume that you have performed an installation of the Enterprise Server as instructed in this chapter, and that all Enterprise Server components reside on the same machine. If you have questions about more complex configurations, please consult your authorized representative for assistance.

For an initial startup, or any time WebLogic Server is not running, follow the steps below:

1. Start any RFID applications that will be communicating with the Enterprise Server.
2. Start the database and application servers:
   a. Start the PointBase server by running the `WL_HOME\common\eval\pointbase\tools\startPointBase` script.
   b. Start WebLogic Server by running the `DOMAIN_HOME\bin\startWebLogic` script.
3. Once WebLogic Server has finished initializing and is in `RUNNING` mode, start any systems that rely on it (for example, the Provisioning Service Administrative Console).

If WebLogic Server is already running, but one or more of the Enterprise Server components are not available, follow the steps below:

1. Use a Web browser to navigate to the WebLogic Administration Console (usually accessible at *http://localhost:7001/console*) and log in using the user name and password you created.
2. Navigate to *Deployments* under your domain in the Domain Structure tree. A Summary of Deployments displays on the right side of the browser window.
3. On the Control tab, select the checkboxes next to the component(s) you want to start. Click **Start** and choose **Servicing all requests** from the drop-down list. Click **Yes** to confirm when prompted.

Stopping Enterprise Server

Before shutting down Enterprise Server components, stop any applications that rely on them (for example, the Provisioning Service Administrative Console).

To stop the Enterprise Server components, follow the steps below:

1. Use a Web browser to navigate to the WebLogic Administration Console (usually accessible at *http://localhost:7001/console*) and log in using the user name and password you created.
2. Navigate to *Deployments* under your domain in the Domain Structure tree. A Summary of Deployments displays on the right side of the browser window.
3. On the Control tab, select the checkboxes next to the component(s) you want to stop. Click **Stop** and choose **When work completes** from the drop-down list. Click **Yes** to confirm when prompted.

To stop the application and database servers:

1. Stop all systems that rely on these servers.
2. Stop WebLogic Server by running the `DOMAIN_HOME\stopWebLogic` script.
3. Stop the PointBase server by running the `WL_HOME\common\eval\pointbase\tools\stopPointBase` script. If you are using another database tool, follow that tool's shutdown instructions.

Chapter 3: Using EPCIS

Contents

This chapter describes how to use the EPCIS component of Enterprise Server.

Overview

The purpose of the EPC Information Service (EPCIS) is to enable enterprise-level EPC-related data sharing and awareness of EPC-tagged objects within a relevant business context. This purpose is accomplished by capturing EPCIS events and making event data available to other enterprise systems. EPCIS events include information about the observed EPCs, such as the relationship between EPCs and EPC locations, as well as the association of those EPCs with business knowledge from your organization.

EPCIS can capture four types of events:

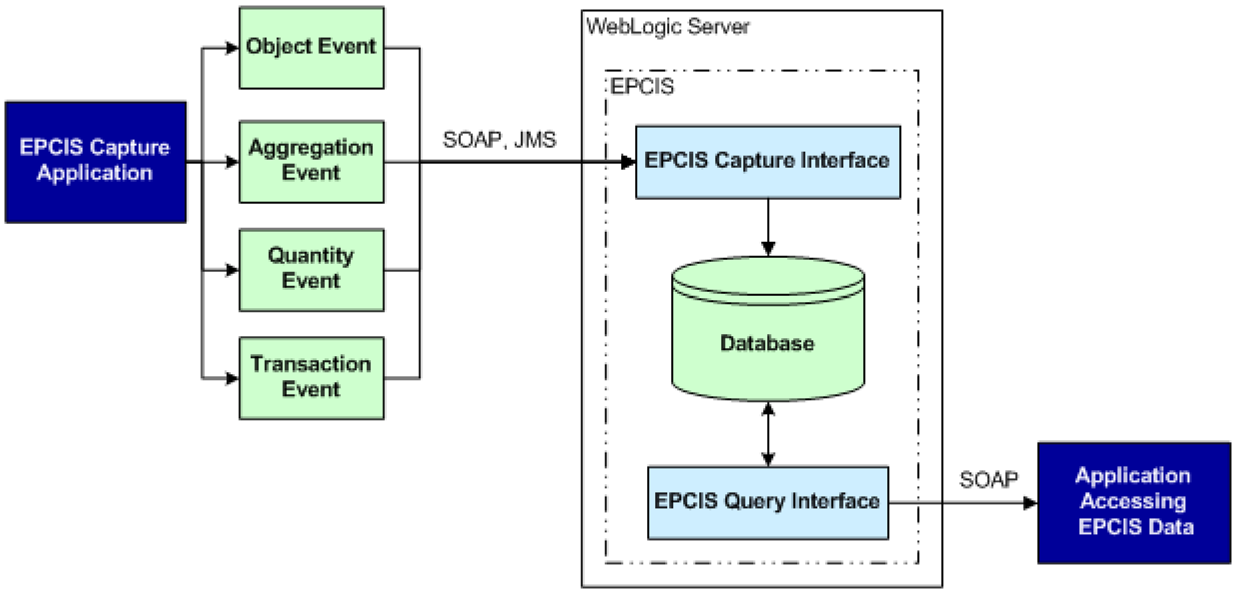
- An *ObjectEvent* captures information about an event pertaining to one or more physical objects identified by EPCs.
- An *AggregationEvent* describes objects that are physically related to one another such that there is a *containing (parent) entity* (for example, a pallet) and a series of *contained (child) objects* (for example, cases on a pallet).
- A *QuantityEvent* captures an event that involves observing a specified quantity of an object class (that is, a particular type of item, such as a bottle containing fifty 200-milligram capsules of a particular pain relief medication).
- A *TransactionEvent* describes the association or disassociation of physical objects to a business transaction.

Each event also contains a notation about what action was taken regarding the EPCs involved in the event. There are three possible actions, each of which may have a slightly different meaning based on the event being captured. These actions are described below.

Action	Meaning
ADD	The entity has been created or added to. For example: • a new EPC commissioned as part of an ObjectEvent • a list of EPCs aggregated to a parent during an AggregationEvent • a list of EPCs associated to a business transaction during a TransactionEvent
OBSERVE	The entity has not been changed (created, added to, destroyed, or removed from), it has simply been observed in transit. For a TransactionEvent, an OBSERVE event confirms the entity's continued association with the transaction.
DELETE	The entity has been removed from or destroyed altogether. For example: • an existing EPC decommissioned as part of an ObjectEvent • a list of EPCs disaggregated from a parent during an AggregationEvent • a list of EPCs disassociated from a business transaction during a TransactionEvent

New EPCIS events are generated at the edge of the enterprise and captured through the EPCIS capture API, where they can subsequently be delivered to interested applications through the EPCIS query API. There is no mechanism to delete or modify an EPCIS event; events are retracted or corrected via a new event whose business meaning is to rescind or amend the effect of a prior event.

Enterprise Server (EPCIS) Components



Effective use of EPCIS requires not only a source of EPC-related data, but also a strictly-controlled vocabulary of business context data. This data is associated with the observed EPC data to form the events sent to EPCIS. It is an important part of the EPCIS data stored in the database. For example, an AggregationEvent contains references to a business location, business process step, business transaction, and business state, in addition to information about the aggregated EPCs. By controlling the vocabulary used for business context data, you can consistently associate that data with EPC observations during event capture. Consistent data also assists applications accessing the EPCIS database to make successful queries.

Integrating EPCIS

This section provides necessary details for developers writing programs that capture EPCIS events from event-generating systems and query stored event information.

Choose one of the following integration scenarios to get started:

- If your organization has (or will have) a system that generates EPCIS events from observing the movements of EPC-tagged objects, you will need to either:
    - Use the EPCIS WSDL interface to develop programs that capture the events and send them (via SOAP) to EPCIS.

- Develop program(s) that capture the events and send XML messages compliant with the EPCIS XSD (via JMS) to EPCIS.

The capture interface provided in the EPCIS API writes the captured events into the EPCIS database, where they are available to other enterprise systems.

See Capture Interface on page 3-4.

Note: It is not within the scope of this document to recommend or prescribe any specific way of enabling your organization's systems to generate EPCIS events.

- If your organization has been capturing EPCIS events using the Enterprise Server and now wants to provide search or query capabilities for the existing data store, you will need to use the EPCIS WSDL interface to develop program(s) that query the event information stored in the EPCIS database via SOAP requests.

  See Query Interface on page 3-5.

Using the EPCIS API

The methods described in this section make up the public SOAP API for the Enterprise Server. There is one method for capturing EPCIS events, and methods for retrieving each of the supported EPCIS events.

In all API calls, EPCs should be provided in Pure Identity format; if EPCs are specified in tag format, they will be stored in tag format and Pure Identity format.

Capture Interface

The CaptureEPCISEvent method captures EPCIS events of four types: Object, Aggregation, Quantity, and Transaction. These event types are described on page 3-2. Captured events are stored in the database.

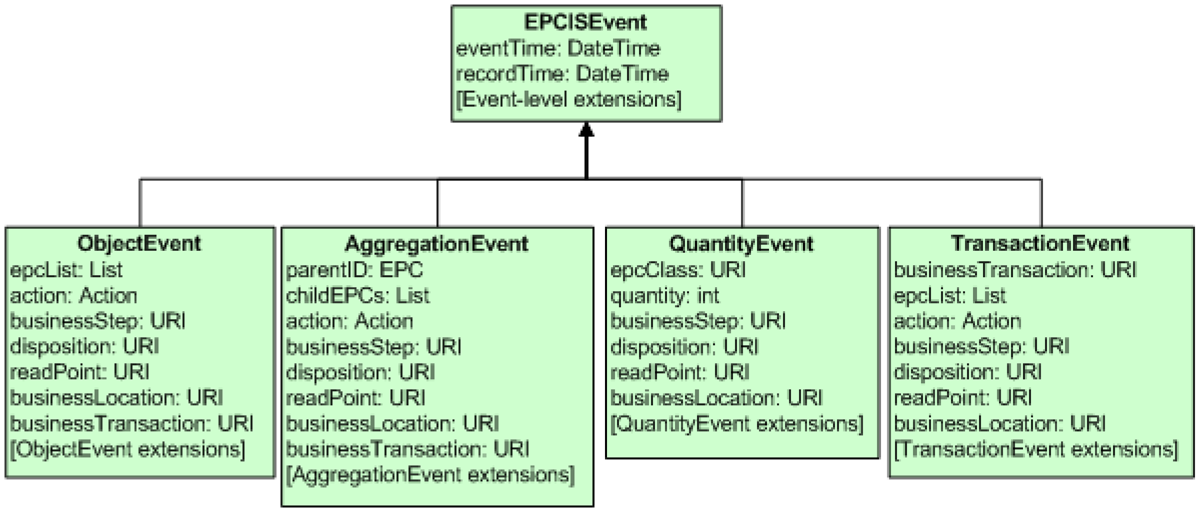
```
captureEPCISEvent(
      epcisEvent : EPCISEvent) : void
```

The `captureEPCISEvent` method takes one argument, an `EPCISEvent`. This data type contains:

- The date and time that the event happened (`eventTime`)
- The date and time the event was recorded (`recordTime`), which must be left blank
- The fields for the specific event type being captured, as shown in the image below

Note: `EPCISEvent` is abstract; only events of the four types shown below will be captured.

EPCIS Events



All the events shown above can be extended with site-specific data; extension data is modeled as a hierarchy of name-value pairs. The EPCIS API supports both capture and query of extension data; these fields appear after the standard object fields. Refer to the WSDL file in `EPCIS_HOME\etc` for more information.

Events can also be captured via JMS rather than SOAP; in this case the event data is directed to the JMS Topic `EPCISTopic` that was set up in Configure WebLogic Server on page 2-8.

The capture interface may throw an `EPCISException`, which indicates an error in processing an EPCIS capture request. See the exception message for details. JMS exceptions typically appear in the WebLogic Server console window, while SOAP exceptions are returned to the calling application.

Query Interface

There are five methods for querying the EPCIS database, one for each of the event types captured by EPCIS, and a generic query that can return multiple event types. To use the query interface, write code that calls the method, filling in the method arguments to create a query. The arguments will be one of the following data types:

- `Date`, in the form `CCYY-MM-DDThh:mm:ss[Z|(+|-)hh:mm]`, where the date is `CCYY.MM.DD`, `T` is a literal, and the time is `hh.mm.ss`. The timezone is either `Z` (UTC time), plus or minus a specified number of hours and minutes, or unspecified. See the following valid examples:

```
2001-10-26T21:32:52
2001-10-26T21:32:52+02:00
2001-10-26T19:32:52Z
2001-10-26T19:32:52+00:00
```

- `List`, usually a comma-separated list of EPC URIs in Pure Identity format as specified by the *EPC Tag Data Standards* (that is, the URI starts with `urn:epc:id:<tag type>:`)

- `String`
- `URI`, a string containing information required to uniquely identify a specific entity

The query arguments given must match the `Date`, `String`, and `URI` data types exactly. For the `List` data type, only one element of a given List needs to match.

When the code is executed, EPCIS queries the database and returns a list of all EPCIS records that match the arguments provided with the query. If an argument if not specified, then it is not used to filter the returned records.

The query interface may throw two kinds of exceptions:

- The `EPCISException` indicates an error in processing an EPCIS query request. See the exception message for details.
- The `QueryTooLargeException` indicates that the number of records returned exceeds the maximum set in the EPCIS configuration. Queries can potentially request large amounts of data. To avoid this exception, you will need to narrow the scope of the query, or configure EPCIS to return more data. Change the following parameter in the `ENTERPRISE_HOME\deploy\epcis.ear\epcis-1.0.war\WEB-INF\epcis-server.props` file:

  `com.connecterra.epcis.query.maxEPCISEventRecords=1000`

GetObjectEvents

ObjectEvents capture event information pertaining to one or more physical objects identified by EPCs. This method retrieves a list of ObjectEvents based on the parameters passed to the function. A null value passed in for any of the parameters means that no filter will be applied to the associated value. URIs must match exactly.

The `getObjectEvents()` call takes eleven arguments:

```
getObjectEvents(
       eventTimeStart : Date,
       eventTimeEnd : Date,
       recordTimeStart : Date,
       recordTimeEnd : Date,
       epcs : List,
       action : String,
       readPoint : URI,
       bizStep : URI,
       businesLocation : URI,
       bizTransaction : URI,
       disposition : URI) : List
```

These arguments are explained in the following table:

Argument	Description
eventTimeStart	Beginning of the time range during which the event happened, inclusive.
eventTimeEnd	End of the time range during which the event happened, inclusive.

Argument	Description
recordTimeStart	Beginning of the time range during which the event was recorded, inclusive.
recordTimeEnd	End of the time range during which the event was recorded, inclusive.
epcs	List of EPC objects in pure identity URI format; records returned match any of the EPCs listed.
action	One of ADD, DELETE, or OBSERVE. See page 3-2 for more information.
readPoint	Location that identifies how or where the event was detected.
bizStep	Business context of the event; the business process step that was happening when the event was captured.
bizLocation	Uniquely identified place where the object(s) observed will be located after the event.
bizTransaction	Business transaction participated in by the event.
disposition	Business state of the object participating in the event (*e.g.*, "available for sale").

GetAggregationEvents

AggregationEvents describe objects that have been physically aggregated to one another such that there is a containing (parent) entity and a series of contained (child) objects. This method retrieves a list of AggregationEvents based on the parameters passed to the function. A null value passed in for any of the parameters means that no filter will be applied to the associated value. URIs must match exactly.

The `getAggregationEvents()` call takes twelve arguments:

```
getAggregationEvents(
      eventTimeStart : Date,
      eventTimeEnd : Date,
      recordTimeStart : Date,
      recordTimeEnd : Date,
      parentEPCs : List,
      childEPCs : List,
      action : String,
      readPoint : URI,
      bizStep : URI,
      bizLocation : URI,
      bizTransaction : URI.
      disposition : URI) : List
```

These arguments are explained in the following table:

Argument	Description
eventTimeStart	Beginning of the time range during which the event happened, inclusive.
eventTimeEnd	End of the time range during which the event happened, inclusive.
recordTimeStart	Beginning of the time range during which the event was recorded, inclusive.
recordTimeEnd	End of the time range during which the event was recorded, inclusive.

Argument	Description
parentEPCs	One or more containing EPC objects in pure identity URI format; records returned match any of the EPCs listed.
childEPCs	List of contained EPC objects in pure identity URI format; records returned match any of the EPCs listed.
action	One of ADD, DELETE, or OBSERVE. See page 3-2 for more information.
readPoint	Location that identifies how or where the event was detected.
bizStep	Business context of an event; the business process step that was happening when the event was captured.
bizLocation	Uniquely identified place where the object(s) observed will be located after the event.
bizTransaction	Business transaction participated in by the event.
disposition	Business state of the object(s) participating in the event (*e.g.*, "available for sale").

GetQuantityEvents

QuantityEvents capture event information that involves observing a specified quantity of an object class. This method retrieves a list of QuantityEvents based on the parameters passed to the function. A null value passed in for any of the parameters means that no filter will be applied to the associated value. URIs must match exactly.

The `getQuantityEvents()` call takes nine arguments:

```
getQuantityEvents(
       eventTimeStart : Date,
       eventTimeEnd : Date,
       recordTimeStart : Date,
       recordTimeEnd : Date,
       epcClasses : List,
       readPoint : URI,
       bizStep : URI,
       bizLocation : URI,
       disposition : URI) : List
```

These arguments are explained in the following table:

Argument	Description
eventTimeStart	Beginning of the time range during which the event happened, inclusive.
eventTimeEnd	End of the time range during which the event happened, inclusive.
recordTimeStart	Beginning of the time range during which the event was recorded, inclusive.
recordTimeEnd	End of the time range during which the event was recorded, inclusive.
epcClasses	One or more EPC objects in EPC ID pattern URI format. Example: `urn:epc:idpat:sgtin:0614141.100734.*` See EPC Identity Patterns on page 4-10 for more information.
readPoint	Location that identifies how or where the event was detected.

Argument	Description
bizStep	Business context of an event; the business process step that was happening when the event was captured.
bizLocation	Uniquely identified place where the object(s) observed will be located after the event.
disposition	Business state of the object participating in the event (*e.g.*, "available for sale").

GetTransactionEvents

TransactionEvents describe the association or disassociation of physical objects to a business transaction. This method retrieves a list of TransactionEvents based on the parameters passed to the function. A null value passed in for any of the parameters means that no filter will be applied to the associated value. URIs must match exactly.

The `getTransactionEvents()` call takes eleven arguments:

```
getTransactionEvents(
       eventTimeStart : Date,
       eventTimeEnd : Date,
       recordTimeStart : Date,
       recordTimeEnd : Date,
       epcs : List,
       action : String,
       readPoint : URI,
       bizStep : URI,
       bizLocation : URI,
       bizTransaction : URI,
       disposition : URI) : List
```

These arguments are explained in the following table:

Argument	Description
eventTimeStart	Beginning of the time range during which the event happened, inclusive.
eventTimeEnd	End of the time range during which the event happened, inclusive.
recordTimeStart	Beginning of the time range during which the event was recorded, inclusive.
recordTimeEnd	End of the time range during which the event was recorded, inclusive.
epcs	List of EPC objects that are part of the transaction (in pure identity URI format); records returned match any of the EPCs listed.
action	One of ADD, DELETE, or OBSERVE. See page 3-2 for more information.
readPoint	Location that identifies how or where the event was detected.
bizStep	Business context of an event; the business process step that was happening when the event was captured.
bizLocation	Uniquely identified place where the object(s) observed will be located after the event.
bizTransaction	Business transaction participated in by the event.
disposition	Business state of the object participating in the event (*e.g.*, "available for sale").

GetEPCISEvents

For queries that return multiple event types, use the generic event query method: `GetEPCISEvents`. This method retrieves a list of events based on the parameters passed to the function. A null value passed in for any of the parameters means that no filter will be applied to the associated value. URIs must match exactly.

The `GetEPCISEvents` call takes thirteen arguments.

```
getEPCISEvents(
       eventTypeArray : List,
       eventTimeStart : Date,
       eventTimeEnd : Date,
       recordTimeStart : Date,
       recordTimeEnd : Date,
       parentEPCs : List,
       childEPCs : List,
       action : String,
       readPoint : URI,
       bizStep : URI,
       bizLocation : URI,
       bizTransaction : URI,
       disposition : URI) : List
```

These arguments are explained in the following table:

Argument	Description
eventTypeArray	List of event types to return (ObjectEvent, AggregationEvent, QuantityEvent, TransactionEvent). If nothing is specified here, all event types are queried.
eventTimeStart	Beginning of the time range during which the event happened, inclusive.
eventTimeEnd	End of the time range during which the event happened, inclusive.
recordTimeStart	Beginning of the time range during which the event was recorded, inclusive.
recordTimeEnd	End of the time range during which the event was recorded, inclusive.
parentEPCs	One or more containing EPC objects in pure identity URI format; records returned match any of the EPCs listed.
childEPCs	List of contained EPC objects in pure identity URI format; records returned match any of the EPCs listed.
action	One of ADD, DELETE, or OBSERVE. See page 3-2 for more information.
readPoint	Location that identifies how or where the event was detected.
bizStep	Business context of an event; the business process step that was happening when the event was captured.
bizLocation	Uniquely-identified place where the object(s) observed will be located after the event.
bizTransaction	Business transaction participated in by the event.
disposition	Business state of the object(s) participating in the event (*e.g.*, “available for sale”).

Chapter 4: Using the Provisioning Service

Contents

This chapter describes how to configure and use the Provisioning Service component of Enterprise Server.

Overview

The purpose of the Provisioning Service component of the Enterprise Server is to automatically provision blocks of Electronic Product Codes (EPCs) to requesting systems. In many cases it is desirable for a tag writing system to write multiple tags using a set of EPCs maintained locally, but replenished from a central repository as more EPCs are needed. These use cases are handled through the use of *EPC caches* (collections of EPC values). The Provisioning Service replenishes the EPC caches maintained by tag writing systems when requested to do so.

Typically, an organization deploying the Provisioning Service already has one or more tag writing systems that require access to a cache of EPCs. You configure the Provisioning Service by:

- Configuring the Provisioning Service component in WebLogic Server.

  See Configure WebLogic Server (page 2-8).

- Setting up serial number selection rules using the Provisioning Service Administrative Console.

  See Creating Selection Rules on page 4-2.

- Establishing contact between the Provisioning Service component and the tag writing systems in use at your organization. This is usually a two-step process:

  - Configure individual tag writing systems to communicate with the Provisioning Service (thereby transforming them into *EPC Clients*).

  - Use the Provisioning Service API to write code that communicates with the Provisioning Service to check out and check in EPCs for a given EPC Client. See Using the Provisioning Service API on page 4-10 for more information.

    Note that this step is not necessary if your organization uses the Provisioning Service to communicate with other RFID software such as Compliance Jump Start.

Creating Selection Rules

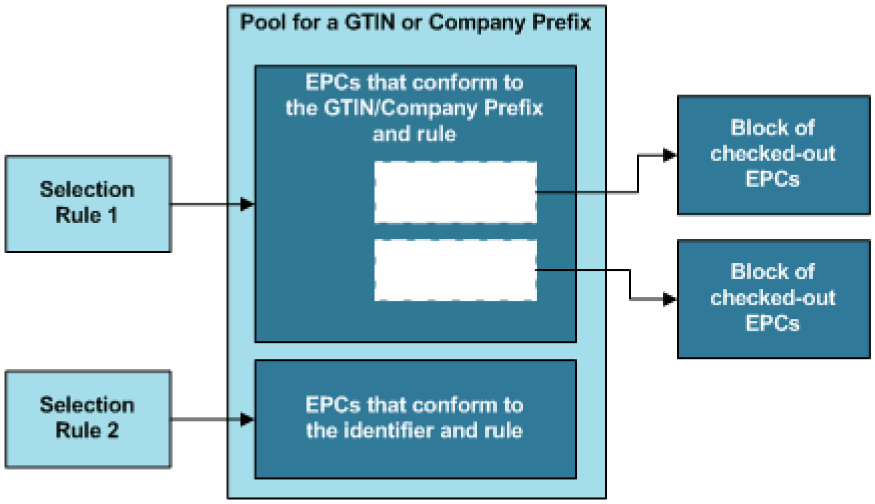
The Provisioning Service can provision two types of EPCs: SGTIN and SSCC. The type of EPC returned depends on the identifier provided by the requesting program (also known as an *EPC Client*). *GTIN*s are provided by the EPC Client when it requests replenishment for a cache of SGTINs. In this case, the Provisioning Service serializes the GTINs to produce SGTIN EPCs. A similar operation occurs when the EPC Client requests replenishment for a cache of SSCCs; the EPC Client provides a *company prefix* value, which is serialized to produce SSCC EPCs. All the available EPCs for a given GTIN or company prefix are called (collectively) a *pool*.

In addition to a GTIN or company prefix, the EPC Client also provides a set of criteria to the Provisioning Service, which uses this information to determine which EPCs will be returned. It does this by matching the criteria received from the EPC Client against the existing selection rules. A *selection rule* is made up of user-defined criteria that define a subset of a pool. If the criteria specified by the EPC Client match the criteria from the selection rule, and it is the *first* matching selection rule, EPCs from that subset of the pool will be returned to the EPC Client. The checked-out EPCs are cached by the EPC Client and are then available for writing onto tags.

The purpose of selection rules is to provide a mechanism for

- Selecting which serial number range within a pool that EPCs are chosen from when responding to a request from an EPC Client.
- Defining subsets of a pool to provide EPCs to various EPC Clients in accordance with business requirements in place at the organization using the Provisioning Service. For example, if your organization has more than one shipping area, you may wish to define selection rules to provide each area with distinct sets of EPCs.

Selection Rules and Pools


Pool for a GTIN or Company Prefix
EPCs that conform to the GTIN/Company Prefix and rule
Selection Rule 1
Block of checked-out EPCs
Block of checked-out EPCs
Selection Rule 2
EPCs that conform to the identifier and rule

When creating selection rules, think about what areas or business units will be requesting EPCs. How many are there? Will each area require separate serial number ranges? Does your corporate data center have any requirements regarding EPC ranges? Is there business logic that needs to be taken into account in making selection rules? By considering these factors, you will be able to set up selection rules that will create EPC pools of adequate size, containing EPCs that provide useful business information to your organization's enterprise systems.

Use the Provisioning Service Administrative Console, accessible via a Web browser, to create and manage selection rules.

Note: Once you log into the Administrative Console, do not use your browser's page controls (such as the **Forward** and **Back** buttons, or bookmarks of Administrative Console pages). Use of these tools may cause unexpected data display errors.

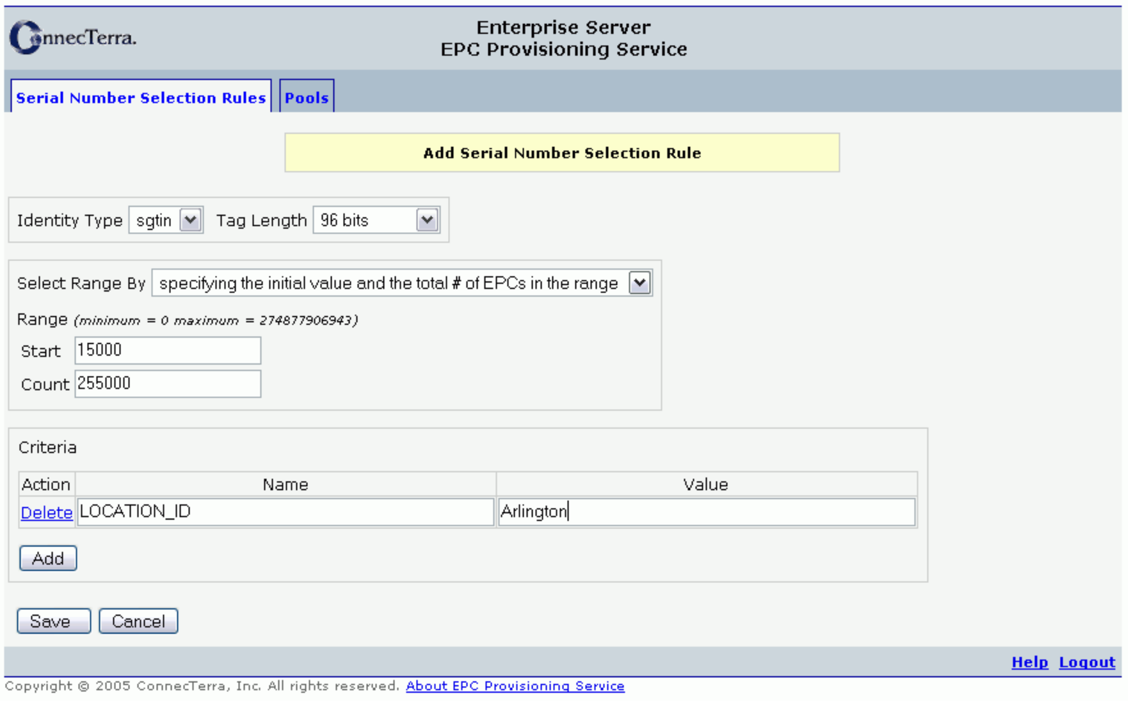
Use the instructions below to view pools and monitor pool usage.

1. Start a Web browser, and navigate to the login page for the Provisioning Service Administrative Console, substituting the machine's hostname for *hostname* in the URL: *http://hostname:7001/epcps_webclient/login.html*

2. Log into the Provisioning Service Administrative Console. (Contact your system administrator if you do not yet have a valid user name.)

3. Click the **Add** button on the Serial Number Selection Rules tab. This action displays the screen shown below.

Add Serial Number Selection Rule Screen



Identity Type: sgtin
Tag Length: 96 bits
Select Range By: specifying the initial value and the total # of EPCs in the range
Range (minimum = 0 maximum = 274877906943)
Start: 15000
Count: 255000

Criteria

Action	Name	Value
Delete	LOCATION_ID	Arlington

Copyright © 2005 ConnecTerra, Inc. All rights reserved. About EPC Provisioning Service

4. Define a selection rule by filling in the fields:

   - *Identity Type* - The type of EPC affected by this rule.
   - *Tag Length* - The kind of tag affected by this rule (96 bits, 64 bits, or Unspecified). Contains the length (in bits) of the EPCs to be returned by the Provisioning Service. If the tag length is Unspecified, 96-bit tags will be returned in response to checkout requests.
   - *Range* - The range of serial numbers affected by this rule. You can specify the range by start/end numbers, by number of EPCs needed, or by using the maximum range possible. After making your choice from the drop-down list, fill in the range fields that display.
   - *Criteria* - The types of criteria (in *Name/Value* pairs) entered for a rule depend on the needs of the business unit using the Provisioning Service. You can use criteria to identify a tagging station, a work unit, a specific GTIN or company prefix, or an area of a warehouse, among other things. Use the **Add** button to add more criteria to the list, if

necessary. Note that the term `TAG_LENGTH` is used internally by the Provisioning Service Administrative Console, and should not be part of any criteria entered into the system.

5. Click **Save** to save the selection rule.

Once you have created selection rules, use the controls to the left of the rule on the Serial Number Selection Rules tab to control rule position and edit or delete a rule.

- Use the **Up** and **Down** buttons next to the rules on the Serial Number Selection Rules tab to put rules in the order you prefer. The order that rules are in matters; a checkout request will use the first rule from the list that matches. Put more specific rules first, leave more general rules until the end of the list. If a more general rule is positioned at the start of the list, it will be used in preference to a more specific rule later in the list.

- To edit a rule, click the **Edit** button next to the rule.

- To delete a rule, click the **Delete** button ☒ next to the rule.

Integrating with Tag Writing Systems

This section provides details that are necessary to the efforts of developers writing programs to integrate the Provisioning Service with existing tag writing systems. Choose one of the following integration scenarios to get started. For scenarios 1 and 3, development can be done in any language that supports classes; this document describes Java classes.

1. If your organization has RFTagAware 1.3.0 or later:

   Set up the RFTagAware Edge Server to send an `EPCCacheReport` (essentially a replenishment request) when the tag cache is running low. Use the Provisioning Service WSDL interface to develop program(s) that receive and act on the `EPCCacheReport`, and use ALEPC API calls to replenish the cache. See RFTagAware on page 4-7 for more information.

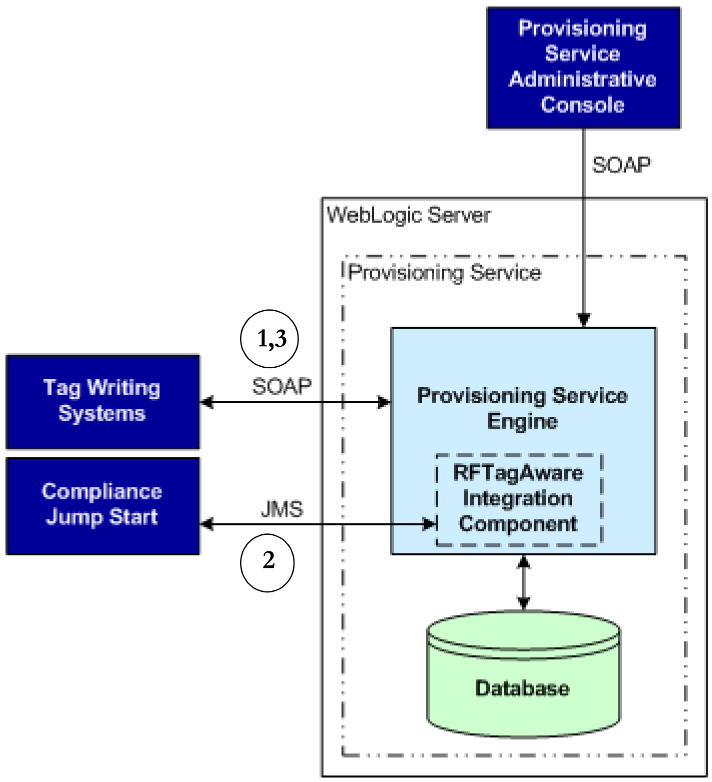
2. If the organization has some other RFID software (Compliance Jump Start, for example):

   Simply configure the software to interact with the Provisioning Service. See Compliance Jump Start on page 4-6 for more information.

3. If your organization has a tag writing system of its own:

   Use the Provisioning Service WSDL interface to develop program(s) that detect when the cache of EPCs kept by a tag writing system is about to be depleted, use the Provisioning Service API to check in and check out EPCs as needed, and return those EPCs to the tag writing system to replenish its EPC cache. See Using the Provisioning Service API on page 4-10 for more information on the Provisioning Service API.

Provisioning Service Components


Provisioning Service Administrative Console
SOAP
WebLogic Server
Provisioning Service
1,3
SOAP
Tag Writing Systems
Provisioning Service Engine
RFTagAware Integration Component
Compliance Jump Start
JMS
2
Database

Compliance Jump Start

To integrate the Provisioning Service with Compliance Jump Start, uncomment and set values for the following properties in the `COMPLIANCE_HOME\etc\tagging-station.props` file.

- The `serviceURL` and `notificationURI` properties give information about how to contact the Provisioning Service.

  ```
  com.connecterra.compliance.epcps.serviceURL=
    http://<ProvisioningService_host>:6065/axis/services/EPCProvisioningService
  ```

  ```
  com.connecterra.compliance.epcps.notificationURI=
    jms:///queue/weblogic.jms.ConnectionFactory/epcpsBrokerMessages
  ```

- The `namingPropertiesFile` property identifies the location of a `naming.props` file for the JMS connection to the Provisioning Service.

  ```
  com.connecterra.compliance.notificationDriver.jms.default.namingPropertiesFile=
  naming.props
  ```

  The file name may include either an absolute path or a path relative to the location of `tagging-station.props`. See Chapter 2 of the *RFTagAware Deployment Guide* for more information on specifying and using the `naming.props` file.

- The `cacheSize` property specifies the number of EPCs to check out and add to the cache.

  ```
  com.connecterra.compliance.epcps.cacheSize=1000
  ```

- The `cacheLowThreshold` property specifies when a cache should be refilled. When the number of EPCs in the cache reaches this value, a cache low notification will be sent so the cache can be replenished. Changes to this value will take effect after Compliance Jump Start is restarted.

  ```
  com.connecterra.compliance.epcps.cacheLowThreshold=10
  ```

- The `userParameter` property specifies a user-defined parameter and value to be sent with EPC checkout requests. There can be multiple parameters specified. These parameters match criteria set in the selection rules.

  ```
  # com.connecterra.compliance.epcps.userParameter.<parameter1>.value=<value1>
  com.connecterra.compliance.epcps.userParameter.Shift.value=Morning
  ```

  Compliance Jump Start defaults to sending EPC checkout requests with the parameters: `GTIN=`*`<value>`*, `TAG_LENGTH=`*`<value>`*, `FILTER_BIT=`*`<value>`*, and `LOCATION_ID=`*`<value>`*, so you should not specify these parameters.

RFTagAware

In RFTagAware, one or more `PCSpec` instances may be associated with an EPC cache. RFTagAware maintains the defined `PCSpec` instances and their EPC caches as part of its persistent state. Each time a `PCSpec` is activated, it takes the next EPC value from its EPC cache, and attempts to write that to a tag. When multiple `PCSpec` instances share a single cache, each will get a different EPC value each time it is activated. The RFTagAware Edge Server's tag programming facility can issue a *cache report* (called `EPCCacheReport`) when the number of remaining EPC values in a cache drops to (or below) a specified level. These reports are essentially replenishment requests for the cache sending the report.

To integrate the Provisioning Service with systems running the RFTagAware Edge Server, you will need to:

- Set up JMS notification for `EPCCacheReports` in RFTagAware. The Edge Server sends these reports when it detects that the tag cache currently being used to print tags is running low. JMS notification setup is described in Setting Up JMS Notification on page 4-8.
- Write code to detect the receipt of an `EPCCacheReport`.
- Use the WSDL interface included with the Provisioning Service to validate and submit an EPC checkout request to the Provisioning Service. The public API is described in Using the Provisioning Service API on page 4-10.
- Use ALEPC API calls to replenish the EPC cache on the Edge Server using the EPCs returned by the Provisioning Service. Please refer to Chapter 6 of the *RFTagAware Programmer Guide* for more information on the ALEPC API.

Setting Up JMS Notification

Edit the following configuration files to use the JMS notification driver with RFTagAware:

> **Note:** In the steps below, `com.connecterra.ale.notificationDriver` is shortened to `[CCAN]`.

1. In `RFTA_HOME\etc\edge.props`, find the JMS notification driver properties and edit as shown:

   a. Find and uncomment the `[CCAN].jms.class` line.

   ```
   [CCAN].jms.class = com.connecterra.ale.notifytypes.JMSNotificationDriver
   ```

   b. Find and uncomment the `[CCAN].jms.default.namingPropertiesFile` line, then set it to the file name of the JNDI naming properties file. The file name may include either an absolute path or a path relative to the location of `edge.props`.

   ```
   [CCAN].jms.default.namingPropertiesFile = C:\\Program
   Files\\ConnecTerra\\RFTagAware\\1.3.1\\etc\\naming.props
   ```

   c. Find the `[CCAN].jms.default.namingInitialContextClass` line. Setting this optional property configures the JMS notification driver to use the specified class to perform JNDI lookups. The value of this class must be a valid Java class available in the system classpath. The value defaults to `javax.naming.InitialContext`, which is used when this property is not specified. Other values are `javax.naming.directory.InitialDirContext` or `javax.naming.ldap.InitialLdapContext`. See the Javadoc for these classes when deciding which class to use.

2. Edit the `RFTA_HOME\etc\jms.options` file as shown:

   a. Edit the `WEBLOGIC_LIB` environment variable to indicate the current WebLogic Server installation:

   ```
   set WEBLOGIC_LIB=%WL_HOME%\server\lib
   ```

   b. Edit the `JMS_LIB` environment variable to indicate the location of the files provided by WebLogic Server that contain the naming context factory class (previously specified in `edge.props`) and all other classes required by WebLogic Server JMS clients.

   ```
   set JMS_LIB=%WEBLOGIC_LIB%\wljmsclient.jar
   ```

3. Create the `RFTA_HOME\etc\naming.props` file, which is used to initialize the instance of the `javax.naming.Context` class. The name/value of the properties specified in the file are used as Context environment properties. For example:

   ```
   java.naming.provider.url=t3://localhost:7001
   java.naming.factory.initial=weblogic.jndi.WLInitialContextFactory
   ```

   The properties in the `naming.props` file are considered default values, and can be overridden by a notification subscription URI (by adding the equivalent property to the notification URI as a query parameter). See the *RFTagAware Programmer Guide* for instructions on overriding these properties.

EPCCacheReport

Java implementation package: `com.connecterra.alepc.api`

A report sent by the RFTagAware Edge Server that indicates that an EPC cache needs replenishing.

```
applicationData : string
cacheContent : EPCPatterns
cacheName : string
cacheSize : long
date : timestamp
savantID : string
threshold : long
---
```

Field	Description
`applicationData`	String that you set in the `EPCCacheSpec`. See Chapter 6 of the *RFTagAware Programmer Guide* for more information.
`cacheContent`	Describes the remaining content of the EPC cache.
`cacheName`	The name of the EPC cache that this report describes.
`cacheSize`	How many EPC cache entries remain.
`date`	The time the report was generated.
`savantID`	Identifier for the Edge Server that generated this report.
`threshold`	The low-cache reporting threshold defined for the `EPCCacheSpec`.

EPCCacheReport - Example

```
<?xml version="1.0" encoding="UTF-8"?>
<EPCCacheReport date="2005-08-27T18:59:32.890Z"
                savantID="test-edge-server"
                xmlns="http://schemas.connecterra.com/alepc">

    <cacheName>mycache</cacheName>

    <applicationData>cache-specific data goes here</applicationData>

    <cacheSize>10</cacheSize>

    <cacheContent>
        <pattern>urn:epc:pat:sgtin-96:0.0614141.100734.[975-999]</pattern>
    </cacheContent>

    <threshold>2500</threshold>

</EPCCacheReport>
```

Using the Provisioning Service API

The methods described in this section make up the public API for the Provisioning Service. Refer to the Javadoc for more information.

EPC Identity Patterns

The Provisioning Service uses EPC identity patterns to specify sets of EPCs that will be checked out (or checked in, in the case of checkin operations). An EPC identity pattern is a URI-formatted string that denotes a single EPC or set of EPCs. The format will be one of:

```
urn:epc:idpat:sgtin:CompanyPrefix.ItemReference.SerialReference
urn:epc:idpat:sscc:CompanyPrefix.SerialReference
```

The fields `CompanyPrefix`, `ItemReference`, and `SerialReference` correspond to fields of an EPC. In an EPC ID pattern returned from the Provisioning Service, the `SerialReference` field may be:

- a decimal integer, meaning that a matching EPC must have that specific value in the corresponding field; or
- a range denoted like `[lo-hi]`, meaning that a matching EPC must have a value between the decimal integers `lo` and `hi`, inclusive.

The complete syntax is defined by the *EPCglobal Tag Data Standards*, Version 1.3. Please consult that document (available at *http://www.epcglobalinc.org/standards_technology/specifications.html*) for full details.

Here are some examples of EPC ID patterns returned by the Provisioning Service. In these examples, assume that `0614141` is the Company Prefix for Widget Corporation, and `100734` is the Item Reference for its UltraWidget product, and that SGTIN tag encodings are used.

Pattern URNs	Description
`urn:epc:idpat:sgtin:0614141.100734.4000`	Matches the tag for UltraWidget serial number 4000.
`urn:epc:idpat:sgtin:0614141.100734.[100-999]`	Matches any Widget Corporation product whose serial number is between 100 and 999, inclusive.

If, for example, the second EPC ID pattern shown above were checked out, the resulting cache contents would be `urn:epc:pat:sgtin-96:0.0614141.100734.[100-999]`.

A cache contains an ordered list of EPC patterns, each of which represents a range of EPC values ordered by ascending serial number; the EPC cache content is the concatenation of the ranges corresponding to pattern URNs in the list.

Note that the filter value and tag length are included with the EPC ID pattern in the checkout request, and that data is used to form the EPC pattern(s) used in the cache.

checkin

This method checks in a set of EPC ID patterns.

```
void checkin(
    com.connecterra.epcps.axis.schema.StringList checkinTypeEpcPatterns)
```

The `checkin()` call takes one argument – a list of EPC ID patterns in a StringList object. If the `checkin()` call succeeds, the EPC ID patterns are checked in.

A `checkin()` call may throw any of the following exceptions: `ServiceException`, `ProvisioningException`, `InvalidURIException`, or `RemoteException`.

checkout

This method checks out a set of EPC ID patterns.

```
com.connecterra.epcps.axis.schema.StringList checkout(
    com.connecterra.epcps.axis.schema.EPCRequestCriteria checkoutTypeParameters,
    int checkoutTypeChunkSize,
    java.lang.String checkoutTypeRequestorID,
    boolean checkoutTypeAllowSmallerChunks)
```

The `checkout()` call takes four arguments, which are explained in the table below:

Argument	Description
Checkout Parameters	An `EPCRequestCriteria` object, which consists of: • an EPC identity type (`SGTIN` or `SSCC`) • an internal criteria name/value pair, which is `GTIN=`*`<value>`* (for an `SGTIN` identity type) or `COMPANY_PREFIX=`*`<value>`* (for an `SSCC` identity type) • any user-specified criteria name/value pairs (which may be from the selection rule used)
Chunk Size	The number of EPCs that the method should return.
Requestor ID	An identifier for the user or service requesting the checkout.
Allow Smaller Chunks?	A boolean, indicating whether the chunk size can be satisfied by returning EPC ID patterns representing non-contiguous blocks of EPCs.

If the `checkout()` call succeeds, the EPC ID patterns are checked out and returned to the calling program as a `StringList`.

A `checkout()` call may throw any of the following exceptions: `ServiceException`, `PoolLowException`, `ProvisioningException`, or `RequestDeniedException`.

validateEPCRequest

This method checks the validity of a potential `checkout()` request. It checks for a selection rule that satisfies the request, and looks for a GTIN or Company Prefix in the `EPCRequestCriteria` object that is passed in. (See checkout on page 4-11 for a description of the `EPCRequestCriteria` object.)

```
java.lang.String validateEPCRequest(
    com.connecterra.epcps.axis.schema.EPCRequestCriteria validateEPCRequestTypeParams)
```

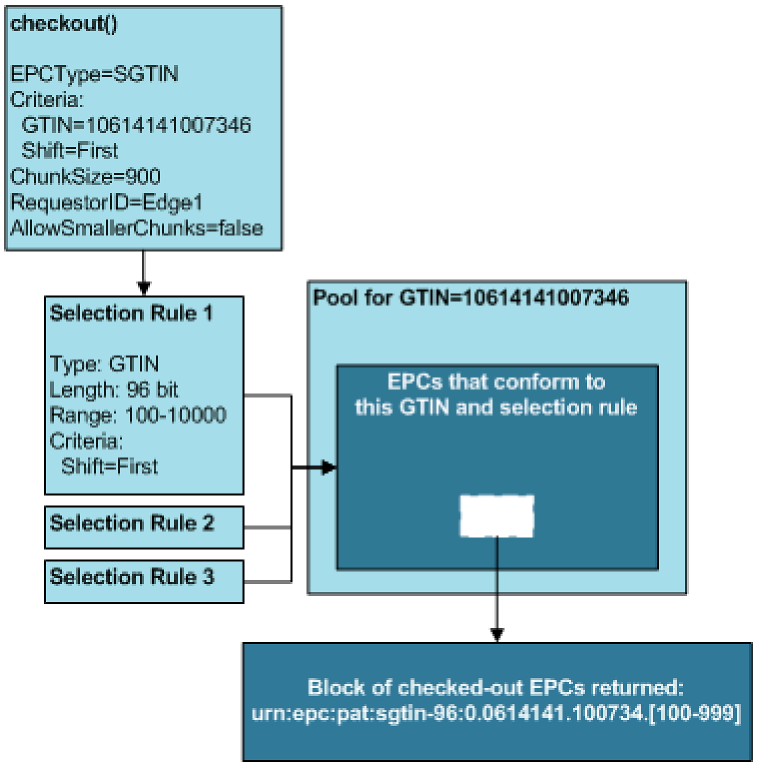
If the `validateEPCRequest()` call succeeds, it returns string representation of the rule that satisfies the request.

A `checkout()` call may throw any of the following exceptions: `ServiceException`, `ProvisioningException`, or `RequestDeniedException`.

Example Checkout Request

The example below shows a `checkout()` request being issued for a particular pool, and a block of checked-out EPCs being returned to the requesting system.

Checkout Request and Response



Monitoring Pools

Once your systems have been integrated with the Provisioning Service, we recommend monitoring all pools for current usage and potential depletion. Use the Provisioning Service Administrative Console, accessible via a Web browser, to perform these monitoring activities.

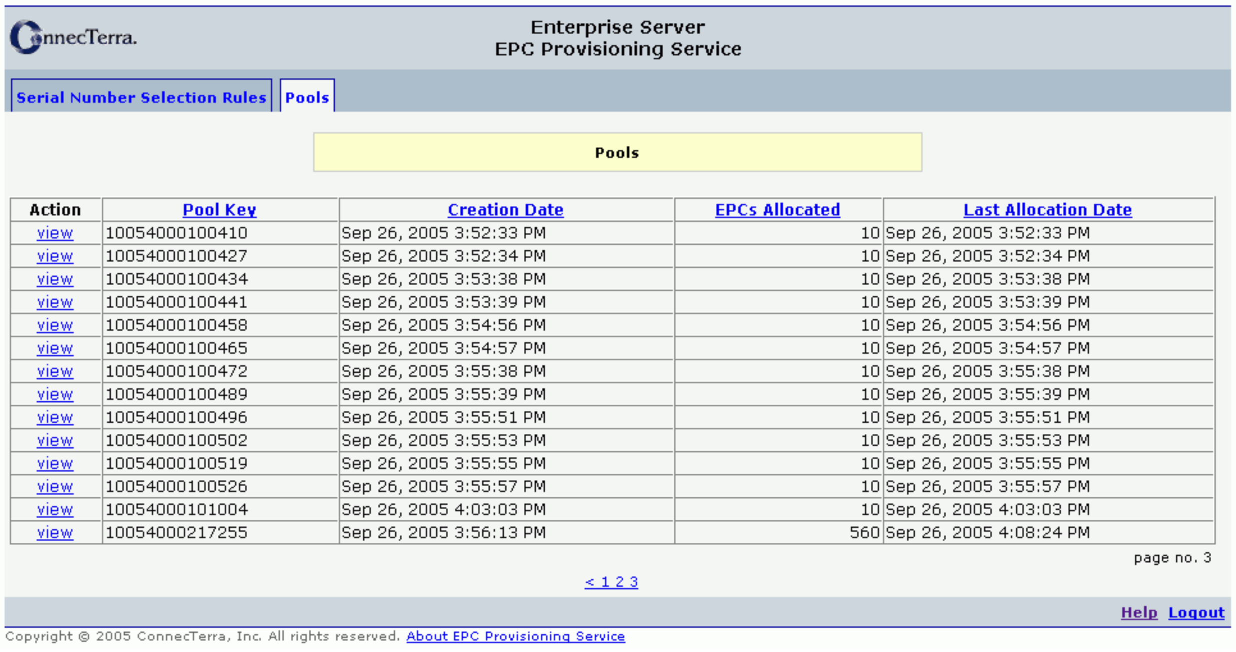
> **Note:** Once you log into the Administrative Console, do not use your browser's page controls (such as the **Forward** and **Back** buttons, or bookmarks of Administrative Console pages). Use of these tools may cause unexpected data display errors.

Use the instructions below to view pools and monitor pool usage.

1. Start a Web browser, and navigate to the login page for the Provisioning Service Administrative Console, substituting the machine's hostname for *hostname* in the URL: *http://hostname:7001/epcps_webclient/login.html*
2. Log into the Provisioning Service Administrative Console. (Contact your system administrator if you do not yet have a valid user name.)
3. Click the Pools tab.

   A list of pools displays. From this page you can monitor all pools in use. The list of pools may be very long. If this is the case, use the page navigation controls beneath the list of pools to access other pages of pool data.

Administrative Console, Pools



Action	Pool Key	Creation Date	EPCs Allocated	Last Allocation Date
view	10054000100410	Sep 26, 2005 3:52:33 PM	10	Sep 26, 2005 3:52:33 PM
view	10054000100427	Sep 26, 2005 3:52:34 PM	10	Sep 26, 2005 3:52:34 PM
view	10054000100434	Sep 26, 2005 3:53:38 PM	10	Sep 26, 2005 3:53:38 PM
view	10054000100441	Sep 26, 2005 3:53:39 PM	10	Sep 26, 2005 3:53:39 PM
view	10054000100458	Sep 26, 2005 3:54:56 PM	10	Sep 26, 2005 3:54:56 PM
view	10054000100465	Sep 26, 2005 3:54:57 PM	10	Sep 26, 2005 3:54:57 PM
view	10054000100472	Sep 26, 2005 3:55:38 PM	10	Sep 26, 2005 3:55:38 PM
view	10054000100489	Sep 26, 2005 3:55:39 PM	10	Sep 26, 2005 3:55:39 PM
view	10054000100496	Sep 26, 2005 3:55:51 PM	10	Sep 26, 2005 3:55:51 PM
view	10054000100502	Sep 26, 2005 3:55:53 PM	10	Sep 26, 2005 3:55:53 PM
view	10054000100519	Sep 26, 2005 3:55:55 PM	10	Sep 26, 2005 3:55:55 PM
view	10054000100526	Sep 26, 2005 3:55:57 PM	10	Sep 26, 2005 3:55:57 PM
view	10054000101004	Sep 26, 2005 4:03:03 PM	10	Sep 26, 2005 4:03:03 PM
view	10054000217255	Sep 26, 2005 3:56:13 PM	560	Sep 26, 2005 4:08:24 PM

page no. 3

< 1 2 3

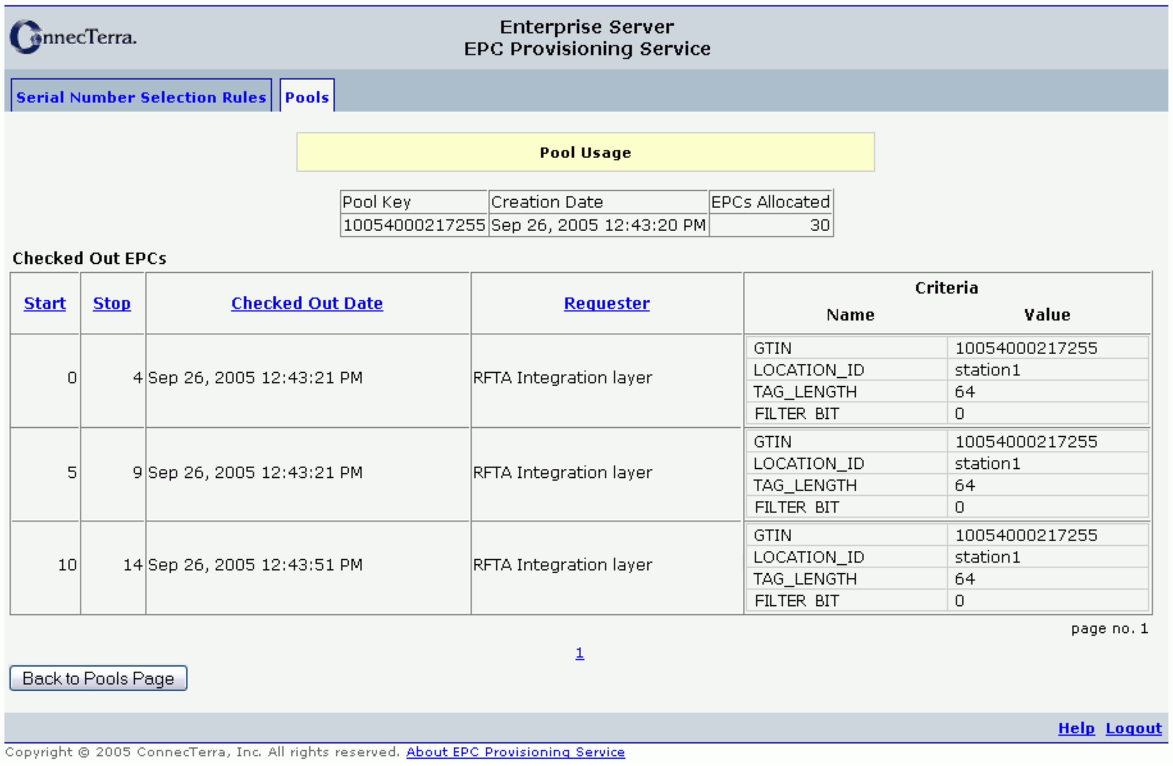
Help Logout

Copyright © 2005 ConnecTerra, Inc. All rights reserved. About EPC Provisioning Service

4. Click the View link next to a specific pool to view pool usage. The Pool Usage page displays.

   A list of checkout times and ranges displays. From this page you can monitor the usage of the pool you chose. Click the **Back to Pools Page** button to return to the original list of pools.

Administrative Console, Pool Usage



Pool Key	Creation Date	EPCs Allocated
10054000217255	Sep 26, 2005 12:43:20 PM	30

Checked Out EPCs

Start	Stop	Checked Out Date	Requester	Criteria Name	Criteria Value
0	4	Sep 26, 2005 12:43:21 PM	RFTA Integration layer	GTIN	10054000217255
				LOCATION_ID	station1
				TAG_LENGTH	64
				FILTER BIT	0
5	9	Sep 26, 2005 12:43:21 PM	RFTA Integration layer	GTIN	10054000217255
				LOCATION_ID	station1
				TAG_LENGTH	64
				FILTER BIT	0
10	14	Sep 26, 2005 12:43:51 PM	RFTA Integration layer	GTIN	10054000217255
				LOCATION_ID	station1
				TAG_LENGTH	64
				FILTER BIT	0

page no. 1

1

Copyright © 2005 ConnecTerra, Inc. All rights reserved. About EPC Provisioning Service

Chapter 5: Data Reference

Contents

This chapter describes the database tables used to hold Enterprise Server data.

EPCIS Database Views

This section documents the public views for data maintained by EPCIS. This information is provided so that you can apply standard database tools to create custom reports or monitor table contents.

To create views for the EPCIS tables, connect to the Enterprise Server database. The `EPCIS_HOME\bin\int-epcis-views.sql` file is an SQL script; run it from the database tool you are using to create the following views:

- epc_view (page 5-2)
- epcis_event_view (page 5-2)
- epcis_event_epc_view (page 5-3)
- epcis_event_extension_view (page 5-3)

epc_view

This view contains one record for every EPC that is recorded as part of an EPCIS event. These records can be shared by multiple EPCIS events. EPC data in this table has been normalized.

Primary Key: `epc_id`

Column Name	Description
company_prefix	Company prefix, derived from the provided EPC URI.
epc_id	The primary key for this table.
filter_value	EPC Filter value, derived from the provided EPC URI.
pure_identity_uri	EPC URI in Pure Identity format, derived from the provided EPC URI.
reference	Item reference, as derived from the provided EPC URI.
scheme	EPC type (SGTIN, SSCC, etc.), derived from the provided EPC URI.
serial_number	Serial number, derived from the provided EPC URI.
uri	EPC URI, provided in the captured event.

Note: If a provided EPC URI is invalid, the fields in this table that are derived from it will be blank for that record.

epcis_event_view

This view contains one record for every EPCIS event captured by the system.

Primary Key: `event_id`

Column Name	Description
action	The action associated with this event.
business_location_uri	The URI for the business location associated with this event.
business_step_uri	The URI for the business step associated with this event.
business_transaction_uri	The URI for the business transaction associated with this event.
disposition_uri	The URI for the disposition of the EPC associated with this event.
event_id	The primary key for this table.
event_time	The time that this event happened.
event_type	The type of event captured (ObjectEvent, AggregationEvent, QuantityEvent, or TransactionEvent).
read_point_uri	The URI of the read point that detected the EPC associated with this event.
record_time	The time that this event was recorded.

epcis_event_epc_view

This view contains records that join EPCIS event data and observed EPCs for each event. It also provides a mechanism to relate parent and child EPCs when necessary (for example, for AggregationEvents).

Primary Key: `event_epc_id`

Column Name	Description
child_epc_id	The EPC associated with this event (from `epc_view`).
child_quantity	The number of EPCs from `epc_view` that match `child_epc_id`. This number will be 1 unless the `child_epc_id` is an EPC ID pattern, which may match 0 or more EPCs.
event_epc_id	The primary key for this table.
event_id	The event identifier (from `epcis_event_view`).
parent_epc_id	The containing or parent EPC, if any.

epcis_event_extension_view

This view contains EPCIS event extension data. Extension data is stored as name-value pairs, with support for hierarchical extensions.

Primary Key: `child_extension_id`

Column Name	Description
child_extension_id	The primary key for this table.
event_id	The event identifier (from `epcis_event_view`).
parent_extension_id	The parent extension, if this is a hierarchical extension.
name	Name of the extension attribute.
value	Value of the extension attribute.

Provisioning Service Database Schema

This section documents the schema for data maintained by the Provisioning Service. This information is provided so that you can apply standard database tools to create custom reports or monitor table contents. You should not attempt to alter information stored in these tables, as they are maintained by the Provisioning Service.

Provisioning Service Tables

allocation_rule

This table contains one record for each selection rule defined via the Provisioning Service.

Primary Key: `allocation_rule_id`

Column Name	Description
identity_type	The type of identifier (SGTIN or SSCC) used for this selection rule.
range_start	The start of the range of EPCs available for this selection rule.
range_end	The end of the range of EPCs available for this selection rule.
ordinal	Order in which the selection rule will be applied.
allocation_rule_id	The primary key for this table.

allocation_rule_criteria

This table stores the criteria name/value pairs referenced in selection rules.

Column Name	Description
allocation_rule_id	The identifier for the selection rule that these criteria are associated with.
parameter_name	Criteria name.
parameter_value	Criteria value.

epc_class

This table stores information about the pools created by applying selection rules. See Creating Selection Rules on page 4-2 for more information.

Primary Keys: `identity_type`, `identity_source`

Column Name	Description
identity_type	The type of identifier (SGTIN or SSCC).
identity_source	The GTIN or Company Prefix used by EPCs from this pool.
num_checkedout_epcs	The total number of EPCs checked out from this pool.
creation_date	Date and time that the pool was created (*i.e.*, when the first checkout happens).
last_checkout_date	Date and time that EPCs were last checked out from the pool.

checkedout_epc_range

This table stores data on checked-out ranges in compacted form, and is maintained for system performance reasons.

Primary Key: `checkedout_range_id`

Column Name	Description
epc_class	The EPC class information, in the form: `sgtin:CompanyPrefix:ItemReference`, or `sscc:CompanyPrefix`
serial_range_start	The starting serial number in this range of EPCs.
serial_range_end	The end serial number in this range of EPCs.
checkedout_range_id	The primary key for this table.

checkout_details

This table contains one record for each checkout performed by the Provisioning Service (that is, pool usage data). Record data includes: who requested the checkout, when the checkout occurred, and the range(s) of EPCs checked out.

Primary Key: `checkout_details_id`

Column Name	Description
epc_class	The EPC class information, in the form: `sgtin:CompanyPrefix:ItemReference`, or `sscc:CompanyPrefix`
serial_range_start	The start of the range of EPCs that has been checked out.

serial_range_end	The end of the range of EPCs that has been checked out.
requestor_id	Identifier of the requestor responsible for the checkout (as given by the requestor).
allocated_date	Date and time these EPCs were checked out.
checkout_details_id	The primary key for this table.

checkout_criteria

This table stores the criteria name/value pairs specified in the Provisioning Service checkout request.

Column Name	Description
checkout_details_id	Unique identifier for a block of checked-out EPCs.
attr_name	Criteria name from the selection rule applied for the checkout.
attr_value	Criteria value from the selection rule applied for the checkout.

Integration Layer Tables

cache_parameters

This table contains the cache parameters for EPC tag caches used by the RFTagAware Integration Layer.

Primary Key: `parameters_id`

Column Name	Description
cache_name	Name of the cache in RFTagAware.
edge_server_id	Identifier for the Edge Server where the cache resides.
cache_id	Unique identifier created by concatenating `cache_name` and `edge_server_id`.
edge_server_url	URL to connect to the ALEPC SOAP interface on this Edge Server. Example: *http://localhost:6060/axis/services/ALEPCService*
identity_type	The type of identifier (SGTIN or SSCC).
cache_filter	Filter value to be used for EPCs in the cache (0-7). Corresponds to the `filter` property in Compliance Jump Start's `tagging-station.props` file. This value is specified by retailer mandate.
cache_tag_size	Length in bits of tags stored in the cache (typically 64 or 96).
cache_fill_size	Number of tags to request when the cache is running low on tags.
parameters_id	The primary key for this table.

cache_selection_criteria

This table stores the criteria name/value pairs used to check out the EPCs for the cache.

Column Name	Description
parameters_id	Unique identifier for the cache that these criteria are associated with.
name	Criteria name from the selection rule used to create this cache.
value	Criteria value from the selection rule used to create this cache.

Chapter 6: Glossary

Action – The type of EPC-related activity observed for a given EPCIS event (limited to ADD, DELETE, or OBSERVE).

Checkout request – A request from an EPC Client that provides a GTIN (or company prefix) and a set of criteria. The Provisioning Service matches the criteria from the checkout request against the criteria from selection rules to determine which EPCs are returned to the requesting EPC Client.

Company prefix – An identifier assigned to a managing entity (company or organization).

Depleted – Having no EPCs in an EPC cache.

EPC – An Electronic Product Code, which uniquely identifies an object and can be used to track it. EPCs are often written to RFID labels.

EPC cache – A set of EPCs that an EPC Client has obtained from the Provisioning Service and not yet assigned to individual products. When the EPC cache runs low, it is replenished by the Provisioning Service upon request. Also called *cache*.

EPC Class – See *EPC ID pattern*.

EPC Client – A software system that uses the Provisioning Service to obtain blocks of EPC serial numbers to assign. EPC Clients are typically deployed in a remote facility.

EPC ID pattern – A URI-formatted string that specifies a set of EPCs of a particular type. EPC ID patterns are used by the Provisioning Service to check EPCs in and out of pools, and by EPCIS to indicate which item is involved in a QuantityEvent. Also called *EPC Class*. **Example:** `urn:epc:idpat:sgtin:0614141.100734.[400-499]`.

EPCIS – The EPC Information Service, a component of the RFTagAware Enterprise Server. It is responsible for capturing EPCIS event information, storing it, and providing access to the stored data.

EPCIS API – The external API used to capture and query EPCIS event data.

EPCIS event – An observation of EPC-related activity, which is usually associated with business processes or transactions.

EPC pool – The set of all possible EPCs for a particular identifier (GTIN or company prefix). Also called *pool*.

GTIN – Global Trade Item Number, a code defined in the EAN.UCC General Specifications identifying a particular class of object such as a product. It is made up of a *company prefix* and an *item reference* (identifier assigned to a particular class of objects or items).

Provisioning Service – A component of the RFTagAware Enterprise Server. It is responsible for managing blocks of EPC serial numbers and allocating (or *provisioning*) them to client software running in remote facilities. The Provisioning Service consists of the Provisioning Service Engine, API, and Administrative Console.

Provisioning Service Administrative Console – The Web interface to the Provisioning Service. System administrators can manage serial number selection rules and view EPC pools and pool usage details using this console.

Provisioning Service API – The external API used by clients to access the Provisioning Service.

Provisioning Service Engine – The component of the Provisioning Service used to implement provisioning service functionality, including defining and maintaining selection rules for allocating EPCs within a pool, and checking out and checking in EPCs in response to requests from EPC Clients.

Replenished – Having at least one EPC in an EPC cache. *Replenishing* is the act of providing EPC values to a cache that is (or is about to be) depleted.

Selection rule – A set of parameters used to define which EPCs should be considered for assignment to a specific EPC Client.

SGLN – Serialized Global Location Number, a standard for identifying unique physical locations. It is made up of a *company prefix* and a self-assigned *location reference*. The serial number field is reserved and should not be used.

SGTIN – Serialized Global Trade Item Number, an EPC identity type. It is made up of a *GTIN* and a *serial number* (unique identifier assigned to an individual object).

SOAP – Simple Object Access Protocol, a protocol for exchanging XML messages, usually via HTTP. See the description available at *http://www.w3.org/TR/soap*.

SSCC – Serial Shipping Container Code, an EPC identity type and code defined in the EAN.UCC General Specifications. It is made up of a *company prefix* and a *serial reference* (identifier assigned to a specific shipping unit).

URI – Uniform Resource Identifier, a unique name that represents the address or location of a resource, typically on the Internet.

WSDL – Web Services Description Language, an XML format for describing Web services. See the description available at *http://www.w3.org/TR/wsdl*.

XML – Exensible Markup Language, a markup language used to describe and transmit data across many different systems. See the description available at *http://www.w3.org/TR/REC-xml*.

Index