Guidewire PolicyCenter®

PolicyCenter System Administration Guide

RELEASE 8.0.3

Guidewire®
Deliver insurance your way™

Product Name: Guidewire PolicyCenter
Product Release: 8.0.3
Document Name: *PolicyCenter System Administration Guide*
Document Revision: 18-November-2014

Contents

About PolicyCenter Documentation

The following table lists the documents in PolicyCenter documentation.

Document	Purpose
InsuranceSuite Guide	If you are new to Guidewire InsuranceSuite applications, read the *InsuranceSuite Guide* for information on the architecture of Guidewire InsuranceSuite and application integrations. The intended readers are everyone who works with Guidewire applications.
Application Guide	If you are new to PolicyCenter or want to understand a feature, read the *Application Guide*. This guide describes features from a business perspective and provides links to other books as needed. The intended readers are everyone who works with PolicyCenter.
Upgrade Guide	Describes how to upgrade PolicyCenter from a previous major version. The intended readers are system administrators and implementation engineers who must merge base application changes into existing PolicyCenter application extensions and integrations.
New and Changed Guide	Describes new features and changes from prior PolicyCenter versions. Intended readers are business users and system administrators who want an overview of new features and changes to features. Consult the “Release Notes Archive” part of this document for changes in prior maintenance releases.
Installation Guide	Describes how to install PolicyCenter. The intended readers are everyone who installs the application for development or for production.
System Administration Guide	Describes how to manage a PolicyCenter system. The intended readers are system administrators responsible for managing security, backups, logging, importing user data, or application monitoring.
Configuration Guide	The primary reference for configuring initial implementation, data model extensions, and user interface (PCF) files. The intended readers are all IT staff and configuration engineers.
Globalization Guide	Describes how to configure PolicyCenter for a global environment. Covers globalization topics such as global regions, languages, date and number formats, names, currencies, addresses, and phone numbers. The intended readers are configuration engineers who localize PolicyCenter.
Rules Guide	Describes business rule methodology and the rule sets in PolicyCenter Studio. The intended readers are business analysts who define business processes, as well as programmers who write business rules in Gosu.
Contact Management Guide	Describes how to configure Guidewire InsuranceSuite applications to integrate with ContactManager and how to manage client and vendor contacts in a single system of record. The intended readers are PolicyCenter implementation engineers and ContactManager administrators.
Best Practices Guide	A reference of recommended design patterns for data model extensions, user interface, business rules, and Gosu programming. The intended readers are configuration engineers.
Integration Guide	Describes the integration architecture, concepts, and procedures for integrating PolicyCenter with external systems and extending application behavior with custom programming code. The intended readers are system architects and the integration programmers who write web services code or plugin code in Gosu or Java.
Gosu Reference Guide	Describes the Gosu programming language. The intended readers are anyone who uses the Gosu language, including for rules and PCF configuration.
Glossary	Defines industry terminology and technical terms in Guidewire documentation. The intended readers are everyone who works with Guidewire applications.

Document	Purpose
Product Model Guide	Describes the PolicyCenter product model. The intended readers are business analysts and implementation engineers who use PolicyCenter or Product Designer. To customize the product model, see the *Product Designer Guide*.
Product Designer Guide	Describes how to use Product Designer to configure lines of business. The intended readers are business analysts and implementation engineers who customize the product model and design new lines of business.

Conventions in This Document

Text style	Meaning	Examples
italic	Emphasis, special terminology, or a book title.	A *destination* sends messages to an external system.
bold	Strong emphasis within standard text or table text.	You **must** define this property.
narrow bold	The name of a user interface element, such as a button name, a menu item name, or a tab name.	Next, click **Submit**.
`monospaced`	Literal text that you can type into code, computer output, class names, URLs, code examples, parameter names, string literals, and other objects that might appear in programming code. In code blocks, bold formatting highlights relevant sections to notice or to configure.	Get the field from the `Address` object.
`monospaced italic`	Parameter names or other variable placeholder text within URLs or other code snippets.	Use `getName(`*`first, last`*`)`. `http://`*`SERVERNAME`*`/a.html`.

Support

For assistance with this software release, contact Guidewire Customer Support:

- At the Guidewire Resource Portal – `http://guidewire.custhelp.com`
- By email – `support@guidewire.com`
- By phone – +1-650-356-4955

chapter 1

Basic Configuration

This topic introduces the `config.xml` configuration file that you use to manage PolicyCenter and describes some initial configuration options for PolicyCenter.

For information about the PolicyCenter installation directory contents, see “PolicyCenter Configuration Files” on page 89 in the *Configuration Guide*.

This topic includes:

The config.xml File

PolicyCenter provides many system parameters to configure its behavior. You set these parameters in the `PolicyCenter/modules/configuration/config/config.xml` file. Access this file from Guidewire Studio under **configuration** → **config**. The `config.xml` file includes PolicyCenter configuration parameters. These parameters govern large-scale system options, such as authentication, application server clustering, the business calendar, default values for business logic rules, and more.

For a list of configuration parameters, see “Application Configuration Parameters” on page 35 in the *Configuration Guide*.

The database-config.xml File

The `database-config.xml` file stores database connection information and Data Definition Language (DDL) options. Access this file from Guidewire Studio under **configuration** → **config**. See "Configuring the Database" on page 27 in the *Installation Guide*.

Defining the Application Server Environment

During startup, PolicyCenter calculates key environment properties. These property values describe the environment in which the application server runs. After you set environment properties, you can access these properties through PolicyCenter commands, Java code, plugins, or Gosu. The environment properties are:

`serverid`	A unique server name or IP address. If you do not specify this value explicitly, PolicyCenter sets it to the `hostname` of the PolicyCenter server. Log entries display only the first 10 characters of the `serverid` value.
`env`	The name of the environment in which the server operates. The default is `null`.
`isbatchserver`	Whether the server is a batch server or not. In a clustered environment, the default value is `false`. In a non-clustered environment, the batch server resolves to `true`.

You can specify environment property values through JVM options or through the `registry` element. Pass JVM options through the command line you use to start the application server. Define the `registry` element in the `config.xml` file. Depending on how many PolicyCenter servers your environment requires, you might find it necessary to adjust the environment properties significantly. You can use environment properties together with the configuration file to specify and control one or multiple application server environments.

If you intend to use clustering in your environment, read this section thoroughly before going on to read "Clustering Application Servers" on page 75. Since `isbatchserver` is only relevant in a clustered environment, a full discussion of that parameter appears in the clustering discussion.

The `gw.api.system.server.ServerUtil` Gosu class contains methods for working with system properties associated with servers. See "Reading System Properties in Plugins" on page 139 in the *Integration Guide* for information on how to use this library.

Setting Java Virtual Machine (JVM) Options

You can use the JVM `-D` flag to specify environment properties for the PolicyCenter server. You can change the environment properties through the `java` command line by specifying any of the following options with the `-D` flag:

- `gw.pc.serverid`
- `gw.pc.env`
- `gw.pc.isbatchserver`

For example, to set the `serverid` property on the command line, specify a `java` command option as follows:

```
java -Dgw.pc.serverid=server name or IP address
```

How to set `java` command options depends on the application server type:

- On JBoss, pass the options as arguments to the JBoss `run` script.
- On Tomcat, set the options in the `CATALINA_OPTS` environment variable.
- On WebLogic, edit the `startManagedWebLogic` file.
- On WebSphere, open the **Administrative Console** and add the option to **Generic JVM arguments**. If you have multiple servers, set the option for each server.

Specifying Environment Properties in the <registry> Element

As an alternative to JVM options, you can specify environment properties by using the `<registry>` element. With this method you can create a single configuration that works on multiple servers. This technique is useful for clustered environments or if you are a member of a development group developing a production configuration.

The `registry` element can contain two subelements: `systemproperty` and `server`. The following example illustrates this element in use:

```
<registry>
  <systemproperty name="env" value="my.env" default="production"/>
  <systemproperty name="serverid" value="my.id" default="mydefault"/>

  <server env="null" serverid="devserver" />
  <server env="test" serverid="testserver" />
  <server env="production" serverid="prodserver" isbatchserver="true"/>

</registry>
```

Use of the `registry` element is optional.

<systemproperty> subelement

Use `systemproperty` elements to rename the `gw.pc.*` Java system properties and set default values. This element has the following attributes:

`default`	Default property value if you do not specify a value on the command line. PolicyCenter requires this attribute.
`name`	Specifies the property that you want to define. This value can be one of: • `env` • `isbatchserver` • `serverid` PolicyCenter requires this attribute.
`value`	Renames the Java option. PolicyCenter requires this attribute.

This value overrides the name of a default `gw.pc.*` option. For example, if you define the following:

```
<systemproperty name="env" value="my.env" default="defaultenv"/>
```

Then, in the JVM options, you specify `-Dmy.env` instead of the `-Dgw.pc.env` option. You do not have to specify any `systemproperty` elements. They are all optional.

<server> subelement

A `server` subelement describes a server instance. This subelement contains the following attributes:

`env`	Specifies the environment in which the server is active. This attribute is optional.
`isbatchserver`	Specifies whether the server is or is not a batch server. This is a Boolean value. This attribute is optional.
`serverid`	References the `serverid` value in which the server is active. This attribute is optional. Log entries display only 10 characters of the `serverid` value.

You can specify multiple `server` elements. The `server` element is optional.

Calculating Environment Property Values

During startup, PolicyCenter calculates environment properties. This section describes how PolicyCenter calculates each environment property.

A `systemproperty` subelement resets the name of the default JVM option. If you specify a property with both a `gw.pc.*` JVM option and a `systemproperty` subelement, PolicyCenter ignores the JVM option. For example, if you specify a `-Dgw.pc.env="test"` JVM option and set the following:

```
<systemproperty name="env" value="my.env" default="standalone" />
```

PolicyCenter ignores any `-Dgw.pc.env` option you specify on the command line and sets the `env` value to `standalone`.

env Property

If you specify the `-Dgw.pc.env` JVM option, PolicyCenter sets `env` to that value. Alternatively, you can define an `env` environment property and set a default value. The following example shows how to set the `env` property in the `registry` element:

```
<registry>
  <systemproperty name="env" value="my.env" default="production"/>
  <systemproperty name="serverid" value="my.id" default="mydefault"/>

  <server env="production" serverid="prodserver" isbatchserver="true"/>

</registry>
```

If you specify the `-Dmy.env=test` option, PolicyCenter sets the `env` to the `test` value. If you do not specify the option, PolicyCenter sets the `env` to the `default` value you specified in the `systemproperty`, in this example, `production`. If the you do not set the `env` either through a default property or with a JVM option, PolicyCenter sets the `env` to `null`.

Note: The `env` property has special significance for logging behavior. If the `env` value is non-null, PolicyCenter tries to obtain the logging configuration from a `config/logging/`*`env`*`-logging.properties` file. If this file does not exist or if `env` is `null`, then the logging configuration is taken from the default `logging.properties` file.

isbatchserver Property

Define at least one server as a batch server. A batch server is a server on which batch processes run.

In a non-clustered environment PolicyCenter initializes the `isbatchserver` environment property to `true` and acts as a batch server. You do not need to set this property explicitly.

In a clustered environment, the `isbatchserver` property must resolve to `true` on at least one server. For a discussion of how the `isbatchserver` property acts in a clustered environment, see “Defining a Batch Server with the isbatchserver Environment Property” on page 79.

For more information on batch processes and work queues that distribute batch processing across multiple servers see “Batch Processing” on page 99.

serverid Property

If you specify the `-Dgw.pc.serverid` JVM option, PolicyCenter sets the `serverid` to that value. Alternatively, you can define the `serverid` value in the `registry` element, as shown in the following example:

```
<registry>
  <systemproperty name="env" value="my.env" default="production"/>
  <systemproperty name="serverid" value="gw.pc.serverid" default="dev1"/>

  <server env="production" serverid="prodserver" isbatchserver="true"/>

</registry>
```

PolicyCenter determines the value of `serverid` during startup. The `serverid` is immutable while the server is running. PolicyCenter determines the value of `serverid` as follows:

1. If you specify the JVM option `-Dgw.pc.serverid` (or the `value` of the `serverid` property defined in a `<systemproperty>`), PolicyCenter uses that value for the `serverid`. For example, while starting the application server, include `-Dgw.pc.serverid=prod1` to set the `serverid` to `prod1`.
2. If you do not specify the JVM option, PolicyCenter checks for a `serverid` property defined by a `<systemproperty>` entry and uses the value of the `default` attribute. In the previous example, the default is `dev1`.
3. If you do not specify the JVM option, and no `serverid` property defined by a `<systemproperty>` entry exists, PolicyCenter sets `serverid` to the host name of the computer. Under some extreme security settings this is not available, in which case PolicyCenter sets the `serverid` to `localhost`. If you run multiple servers on the same host computer, define a `serverid` for each server with a `<systemproperty>` entry.

   Note: Log entries display only the first 10 characters of the `serverid` value.

Specifying Parameters by Environment

Typically, you need to support more than one server environment. Guidewire recommends you maintain at least development, test, and deployment environments. To prevent you from having to change `config.xml` parameters each time you switch between environments, PolicyCenter provides configuration parameters that you can set by environment.

Environment-specific parameters can reference environment properties to indicate in which environment they are valid. You specify the environment for a parameter by adding one or both of an `env` or `server` attribute. For example:

```
<param name="BusinessDayStart" value="9:00 AM" server="dev1" />
<param name="BusinessDayStart" value="7:00 AM" env="test" />
<param name="BusinessDayStart" value="8:00 AM"/>
```

Then, you can start the application server and have PolicyCenter use the environment-specific parameter by specifying the environment in JVM options. Continuing the example, to have `BusinessDayStart` resolve to 7:00 AM, specify the `test` environment in your JVM options:

```
-Dgw.pc.env=test
```

If PolicyCenter resolves `serverid` to `dev1`, PolicyCenter sets `BusinessDayStart` to the `9:00 AM` value.

If you define environment-specific parameters, PolicyCenter applies the setting if either the `env` or `server` resolves. For example, if you were to specify the `BusinessDayStart` parameter as follows:

```
<param name="BusinessDayStart" value="9:00 AM" env="test" server="prodserver"/>
<param name="BusinessDayStart" value="8:00 AM"/>
```

PolicyCenter sets `BusinessDayStart` to `9:00 AM` if either `env` resolves to `test` or `serverid` resolves to `prodserver`. For example, if PolicyCenter resolves `env` to `test` and `serverid` to `chicago`, the `BusinessDayStart` is `9:00 AM`. Similarly, if PolicyCenter resolves `env` to `production` and `serverid` to `prodserver`, the `BusinessDayStart` is also `9:00 AM`. If `env` does not resolve to `test` and `server` does not resolve to `prodserver`, PolicyCenter uses the default `BusinessDayStart` of `8:00 AM`.

For a list of configuration parameters, including information about which parameters can be set by environment, see “Application Configuration Parameters” on page 35 in the *Configuration Guide*.

Providing Default Values

At startup, PolicyCenter requires many parameters. Consider this carefully if you specify parameters by environment. In some cases, you might want to specify a parameter without any `env` or `server` attribute to assure the parameter always resolves to some value. In the following example, the last line always resolves if the other two values do not:

```
<param name="ClusteringEnabled" value="false" server=”chicago” />
<param name="ClusteringEnabled" value="true" env=”test” />
<param name="ClusteringEnabled" value="true"/>
```

The last line setting in this example acts as the default value for the parameter. Of course, you might want the server to start only if a certain environment is available. In this case, a default is inappropriate.

Special Environment-specific Parameters

The `database`, and `plugin` elements are special cases of environment-specific parameters. For the database, you can only specify an `env` attribute. For example, you can connect to different database instances, depending whether you work in the test or development environment. To connect to two different instances, define two database elements in `config.xml`, as shown in the following example:

```
<database name="PolicyCenterDatabase"
  driver="dbcp"
  dbtype="sqlserver"
  autoupgrade="false"
  checker="false"
  env="development">
  <param name="jdbcURL"
   value="jdbc:microsoft:sqlserver://localhost:1433;DatabaseName=pcDev;
   User=sa;Password=123" />
   ...
</database>

<database name="PolicyCenterDatabase"
  driver="dbcp"
  dbtype="sqlserver"
  autoupgrade="false"
  checker="false"
  env="test">
  <param name="jdbcURL"
   value="jdbc:microsoft:sqlserver://localhost:1433;DatabaseName=pcTest;
   User=sa;Password=123" />
   ...
</database>
```

For the `plugin` parameters, you can specify one or both of the `env` and `server` attributes. A plugin is only active in the following cases:

- Both `env` and `server` attributes are present and both reference a resolved environment property.
- Only an `env` or a `server` attribute is present and it references a resolved environment property.
- Neither attribute is present.

Using the Geocoding Feature

Guidewire supports *geocoding* within PolicyCenter and ContactManager. The geocoding process assigns latitudes and longitudes to addresses. Software then uses geocoded addresses to present users with geographic information, such as the distance between two addresses. All primary addresses in PolicyCenter and ContactManager are candidates for geocoding.

Working with the Geocode Plugin

To implement geocoding, PolicyCenter provides the `GeocodePlugin`. Implementations of the plugin connect with specific external geocoding services, which provide geocode coordinates for specific addresses. Typically, plugin implementations also use an external mapping service to calculate and return proximity information, driving instructions, and maps. Guidewire Studio lets you enable the `GeocodePlugin`, specify which implementation to use, and specify parameters for the implementation you choose.

PolicyCenter provides a fully functioning and supported `GeocodePlugin` implementation, the `BingMapsPlugin` Gosu class. This plugin implementation connects to the Microsoft Bing Maps Geocode Service. If you intend to use the `BingMapsPlugin` implementation, your organization must have a valid account with Microsoft.

See also

- "Geographic Data Integration" on page 229 in the *Integration Guide*

Working with Geocode Batch Processing

Geocode batch processing is implemented as a work queue. Geocode batch processing searches the database for `Address` instances with the `BatchGeocode` property set to `true` and the `GeocodeStatus` property set to `none`. The geocoding process submits qualifying addresses to the `GeocodePlugin`. After the `GeocodePlugin` adds geocode coordinates to an address, the geocoding process updates the address in the database.

Configuring Geocoding

Configuring geocoding in PolicyCenter involves enabling the `GeocodePlugin`, setting geocoding feature parameters, and scheduling the `Geocode` work queue in PolicyCenter. If you use ContactManager, you must do the same.

Enabling the Geocode Plugin

By default, the `GeocodePlugin` plugin uses the Bing Maps implementation, but the plugin is disabled.

Before you can use the Bing Maps plugin implementation, your organization must have its own account, login, and application key with Bing Maps. For more information, go to `http://www.bingmapsportal.com`, where you can set up a Bing Maps account and obtain an application key. When you create an application key, the application name is arbitrary and no application URL is required.

To enable and use the Bing Maps plugin in PolicyCenter

1. Start PolicyCenter Studio.

   At a command prompt, navigate to `PolicyCenter\bin` and enter the following command:

   ```
   gwpc studio
   ```

2. Expand **configuration** → **config** → **Plugins** and double-click to open `GeocodePlugin.gws`.
3. Click the **Enabled** check box to enable the plugin.
4. Assure the **Class** field specifies the Bing Maps implementation class:

   `gw.plugin.geocode.impl.BingMapsPlugin`

5. Under **Parameters**, specify:

applicationKey	The application key that you obtained from Bing Maps.
geocodeDirectionsCulture	The locale for geocoded addresses and routing instructions returned from Bing Maps. For example, use the locale code `ja-JP` for addresses and instructions for Japan. The plugin uses `en-US` if you do not specify a value. For a current list of codes that Bing Maps supports, see `http://msdn.microsoft.com/en-us/library/cc981048.aspx`.

imageryCulture	The language for map imagery. For example, use the language code `ja` for maps labeled in Japanese. The plugin uses `en` if you do not specify a value. For a current list of codes that Bing Maps supports, see `http://msdn.microsoft.com/en-us/library/cc981048.aspx`.
mapUrlHeight	Height of maps, in pixels. The plugin uses `500` if you do not specify a value.
mapUrlWidth	Width of maps, in pixels. The plugin uses `500` if you do not specify a value.

6. Save your changes.

IMPORTANT If you also use ContactManager, repeat your changes in ContactManager Studio.

Setting Geocoding Feature Parameters

Configure geocoding features in the user interface with the following parameters in `config.xml`.

Parameter	Description	default
`UseGeocodingInPrimaryApp`	If `true`, PolicyCenter enables searching for nearby locations in the Reinsurance Management user interface. ContactManager does not respond to this parameter.	`false`
`ProximitySearchOrdinalMaxDistance`	The maximum distance to use while performing an ordinal (nearest *n* items) proximity search for locations. This distance is in miles, unless `UseMetricDistancesByDefault` is `true`.	`300`
`ProximityRadiusSearchDefaultMaxResultCount`	The maximum number of results to return if performing a radius (within *n* miles or kilometers) proximity search. This parameter has no effect on ordinal (nearest *n* items) proximity searches.	`1000`
`UseMetricDistancesByDefault`	If `true`, PolicyCenter uses kilometers and metric distances instead of miles and United States distances for location searches. Set this parameter identically in both PolicyCenter and ContactManager.	`false`

Scheduling and Configuring Geocode Batch Processing

You must edit the following files to schedule and configure geocode batch processing.

- `scheduler-config.xml`
- `work-queue.xml`

Access `scheduler-config.xml` in Guidewire Studio at **configuration** → **config** → **scheduler.** Access `work-queue.xml` in Guidewire Studio at **configuration** → **config** → **workqueue.**

Scheduling when Geocode Batch Processing Runs

You schedule geocode batch processing by modifying the following section in `scheduler-config.xml` file.

```
<ProcessSchedule process="geocode">
  <CronSchedule hours="1" minutes="30"/>
</ProcessSchedule>
```

By default, geocode batch processing is not scheduled. To schedule geocode batch processing, uncomment the section. By default, the schedule runs geocode batch processing at 1:30 AM daily. If you regularly add many new contacts, especially in ContactManager, tune the schedule to match your expected daily load of new addresses.

IMPORTANT Schedule geocode batch processing for PolicyCenter and ContactManager with sufficient processing windows between runs to assure sufficient time for runs to fully process the work items in the work queues. If you find duplicate work items in the work queues for the same address ID, extend the interval between runs.

Configuring the Number of Worker Instances for Geocode Batch Processing

You configure geocode batch processing by modifying the following section in the `work-queue.xml` file.

```
<work-queue workQueueClass="com.guidewire.pc.domain.geodata.geocode.PCGeocodeWorkQueue"
 progressinterval="600000">
  <worker instances="1"/>
</work-queue>
```

The default configuration specifies one worker instance. Worker instances pass addresses from the work queue to the `GeocodePlugin`. Consider increasing the number of worker instances to improve throughput. To further improve throughput, assign worker instances to run on multiple servers.

Performance Considerations with the Geocoding Process

At the time you first start PolicyCenter, your database might have many addresses to geocode, especially if you imported many new addresses into your production database. If your production database has many new addresses, the `GeocodePlugin` might take a long time to process these new addresses. Configure the geocoding process with a sufficient number of worker instances before you start your production servers. Configure the geocoding process to run during periods of minimal activity on the PolicyCenter servers. The geocoding process can potentially update a large number of `Address` records in the database. After the geocoding process completes, update database statistics by running the `incrementaldbstats` process.

See also

- “Scheduling Work Queues and Batch Processes” on page 107
- “Batch Processing” on page 99
- “Running Batch Processes from the Command Line” on page 112
- “Configuring Database Statistics” on page 42

Geocode Status

The `GeocodeStatus` typelist defines the set of status codes returned from the plugin. This typelist is final, so you cannot edit it. Access the `GeocodeStatus` typelist from Guidewire Studio by opening `GeocodeStatus.tti` in **configuration** → **config** → **Metadata** → **Typelist**.

See also

- “Geocoding Status Codes” on page 237 in the *Integration Guide*

Configuring an Email Server for Notifications

PolicyCenter supports sending email notifications from business rules. Sending email is one of several possible actions to take, for example, if a user escalates an activity.

A rule that sends email provides `To` and `From` properties, a subject, the name of the template used to generate the body, and the object that the message references. PolicyCenter initially saves email messages and then sends them to an SMTP email server in a background process.

Guidewire provides a default email message plugin. To set up PolicyCenter for email notifications, configure the plugin parameters in Guidewire Studio.

To configure email plugin parameters in Guidewire Studio

1. Start PolicyCenter Studio.

   At a command prompt, navigate to `PolicyCenter\bin` and enter the following command:

   ```
   gwpc studio
   ```

2. Expand **configuration** → **config** → **Plugins** → **registry** and double-click to open `emailMessageTransport.gws`.
3. If the **Enabled** checkbox is not selected, select the checkbox to enable the plugin.
4. Edit the values set to the following parameters:
   - `defaultSenderAddress`
   - `defaultSenderName`
   - `smtpHost`
   - `smtpPort`
5. Save your changes.
6. Restart the PolicyCenter server.

See also

- “Document Management” on page 191 in the *Integration Guide*
- “Sending Emails” on page 93 in the *Rules Guide*
- “Using the Messaging Editor” on page 131 in the *Configuration Guide*
- “Using Activity Patterns with Documents and Emails” on page 410 in the *Configuration Guide*

Changing the Unrestricted User

By default, PolicyCenter uses the `su` user as the user with unrestricted access. You can set the unrestricted user to another existing user by specifying the `UnrestrictedUserName` parameter in `config.xml`. To set the unrestricted user, set the `value` of the `UnrestrictedUserName` parameter to the user login name:

```
<param name="UnrestrictedUserName" value="user login name"/>
```

chapter 2

Configuring Logging

This topic discusses logging in PolicyCenter.

This topic includes:

Logging Overview

The `PolicyCenter/modules/configuration/config/logging/logging.properties` file specifies system logging options for the PolicyCenter application server. Access this file from Guidewire Studio in **configuration** → **config** → **logging**.

PolicyCenter uses the Java log4j logging facility. To determine the log4j version, check the filename of the log4j JAR file in `PolicyCenter/lib`.

The `logging.properties` file uses the format specified by `log4j`. The entries in `logging.properties` control what to log and in which file to write the log. The setting that controls basic logging looks like the following:

```
log4j.rootCategory=INFO, Console, DailyFileLog
```

This setting indicates that PolicyCenter send system-wide informational messages to two output points: the `Console` and the `DailyFileLog` file. The `INFO` value is the default logging level.

Within `logging.properties`, entries like `log4j.appender.*` indicate the parameters of each output point. These entries identify properties such as location or output format options. As PolicyCenter starts, it attempts to write a log file in the location specified by `log4j.appender.DailyFileLog.File`. By default, this directory is `tmp/gwlogs/PolicyCenter/logs/pclog.log`. PolicyCenter creates the log file automatically. However, if the directory specified by `DailyFileLog.File` does not exist, PolicyCenter writes log information only to the console.

Notes

- See `http://logging.apache.org/log4j/1.2/index.html` for detailed information about creating log4j entries.
- If the `env` environment property is non-null, PolicyCenter tries to obtain the logging configuration from an *`env`*`-logging.properties` file in `PolicyCenter/modules/configuration/config/logging`. For example, if you have an environment called test, PolicyCenter looks for logging properties in `test-logging.properties`. If this file does not exist, or if `env` is `null`, then the logging configuration is taken from the default `logging.properties` file. See "Defining the Application Server Environment" on page 14 for information on the `env` property.
- After you edit the `logging.properties` file, you must rebuild and redeploy the PolicyCenter application file for the changes to take effect.

Specifying Location of Log Files for the View Logs Page

PolicyCenter includes a log viewer on the **Server Tools** page **View Logs**. The `guidewire.logDirectory` property in `logging.properties` specifies the location of log files for the log viewer. Set `guidewire.logDirectory` to a directory where you store log files that you want to be visible from the **View Logs** page. Ensure that log file locations that you specify with `log4j.appender.`*`category`*`.File` are set to the same directory as `guidewire.logDirectory` for log files that you want visible from the **View Logs** page.

See "View Logs" on page 143.

Configuring Logging in a Multiple Instance Environment

You can use variables to specify log file names and locations. This is particularly useful if there are multiple instances on the same physical server. This enables you to use a common `logging.properties` file and generate log files for each instance.

To configure logging by using variables

1. Define `-Dgw.pc.env` and `-Dgw.pc.serverid` parameters for each server instance. For example:

   ```
   -Dgw.pc.env=prod -Dgw.pc.serverid=prodserver1
   -Dgw.pc.env=prod -Dgw.pc.serverid=prodserver2
   ```

2. In the `logging.properties` file, create a variable for the logging directory. For example:

   ```
   guidewire.logDirectory = /apps/pc/logs
   ```

   Set this variable to an absolute path to a directory that already exists.

3. Use the `guidewire.logDirectory` and `-Dgw.pc.serverid` variables in the value set to `log4j.appender.DailyFileLog.File`:

   ```
   log4j.appender.DailyFileLog.File=${guidewire.logDirectory}/${gw.pc.serverid}-pclog.log
   ```

4. Define the `env` and `serverid` system properties within a `<registry>` block in `config.xml`. For example:

   ```
   <systemproperty name="env" value="gw.pc.env" default="local"/>
   <systemproperty name="serverid" value="gw.pc.serverid" default="localhost"/>
   ```

   See "Specifying Environment Properties in the <registry> Element" on page 15.

5. Add the servers to the `<registry>` block in `config.xml`. For example:

```
<server env="prod" serverid="prodserver1"/>
<server env="prod" serverid="prodserver2"/>
```

When you start the servers, each server writes a log file to the common log file directory you specified. Each log file includes the `serverid`. In this example, the `/apps/pc/logs` directory contains two log files after you start the server: `prodserver1-pclog.log` and `prodserver2-pclog.log`.

Logging Levels

In the `logging.properties` file you must express file locations as an absolute path. Regardless of operating system, you must use forward slashes and not backslashes. The directory path that you specify must exist. PolicyCenter creates the log file itself automatically.

The logging level determines how much information you want to record in the log. PolicyCenter reports several levels of information that are, in order of severity, as follows:

Level	Description
TRACE	Messages about processes that are about to start or that completed in order to provide flow-of-control logging. Trace logging has no or minimal impact on system performance. Typical messages might include: • "Calling plugin." • "Returned from plugin call".
DEBUG	Messages that test a provable and specific theory intended to reveal some system malfunction. These messages need not be details but include information that would be understandable by an administrator. For example, dumping the contents of an XML tag or short document is acceptable. However, exporting a large XML document with no line breaks is usually not appropriate. Typical messages might include: • "Length of Array XYZ = 2345." • "Now processing record with public ID ABC:123456."
INFO	Messages that convey a sense of correct system operation. Typical messages might include: • "Component XYZ started." • "A user logged on to PolicyCenter."
WARN	Messages that indicate a potential problem. Examples include: • "An assignment rules did not end in an assignment." • "Special setting XYZ was not found, so the default value was used." • "A plugin call took over 90 seconds."
ERROR	Messages that indicate a definite problem. Typical messages might include: • "A remote system has refused a connection to a plugin call." • "PolicyCenter can not complete operation XYZ even with a default."

Set the logging level property of the different logging entries to set how much information you want sent to each type of log. Logging levels are inherited from parent categories unless the child category has a different level defined. For example, setting `Messaging` to `DEBUG` also sets `Messaging.Email` and `Messaging.Events` to `DEBUG`, unless those categories have another setting specified. The `RootLogger` category is the parent of all other categories. Setting the logging level on `RootLogger` causes all categories to inherit the same level, provided those categories have not set a different logging level. Refer to log4j documentation for a complete discussion of logging levels and inheritance.

Logging Successfully Archived Policy Terms and Policy Periods

PolicyCenter creates a separate log for successfully archived objects. Each log is unique to a run of the archive work queue. The log contains a list of the policy terms and policy periods that were successfully archived.

To configure the log, modify the archiving properties in `logging.properties`. Uncomment the properties that are commented out in the default `logging.properties`.

```
##### Archived Policies #####
# Set up the logging for Archived Policies.

# Root archiving logger
# log4j.category.Server.Archiving=INFO, ArchivedPoliciesLog

# Logger for successful archived policies
# log4j.category.Server.Archiving.Success=INFO, ArchivedPoliciesLog

# Logger for debugging the domain graphs of policies being archived
# log4j.category.Server.Archiving.Graph=DEBUG, ArchivedPoliciesLog

# Logger for the the archiving upgrade process
# log4j.category.Server.Archiving.DocumentUpgrade=INFO, ArchivedPoliciesLog

log4j.appender.ArchivedPoliciesLog=org.apache.log4j.DailyRollingFileAppender
log4j.appender.ArchivedPoliciesLog.File=/tmp/gwlogs/archivedPolicies.log
log4j.appender.ArchivedPoliciesLog.DatePattern = .yyyy-MM-dd
log4j.appender.ArchivedPoliciesLog.layout=org.apache.log4j.PatternLayout
log4j.appender.ArchivedPoliciesLog.layout.ConversionPattern=%-10.10X{server} %-4.4X{user} %d{ISO8601}
  %p %m%n
```

Logs for Additional System Components

The `logging.properties` file contains additional logging categories for other system components such as plugins or integration code. These entries are, by default, commented out. To enable a logging category, uncomment the appropriate `log4j.category.*` entry. For example, the following line turns on debugging for all integration code:

```
log4j.category.Integration=DEBUG, IntegrationLog
```

Just as with the daily log, the locations for these log files must exist. The most important settings to change are the location of the logs and the logging threshold. For example, the following line directs PolicyCenter to write more detailed logging related to the rule engine to an output point called `RuleEngineLog`:

```
log4j.category.com.guidewire.pc.server.rule=DEBUG, RuleEngineLog
```

Then, you can configure the parameters related to the `RuleEngineLog` appender to set up a log file specifically for troubleshooting rules.

Configuring Information in Log Messages

You can modify the `log4j.appender.`*`log`*`.layout.ConversionPattern` value to change the information included in log messages for a log type. For example, to list the logging category for console logs, add `%c` to the `log4j.appender.Console.layout.ConversionPattern` value. You can then filter logs by category.

The following table lists characters that you can use with log4j to customize logging messages.

Character	Description
%%	Writes the percent sign to output.
%c	Name of the logger category. See “Listing Logger Categories” on page 28 for categories provided with PolicyCenter.
%C	Name of the Java class. Because the PolicyCenter logging API is a wrapper around log4j, `%C` returns the class name of the logger. If you want class names in your log messages, include them specifically in the message rather than by using `%C` in the conversion pattern.

Character	Description
%d	Date and time. Acceptable formats include: • %d{ISO8601} • %d{DATE} • %d{ABSOLUTE} • %d{HH:mm:ss,SSS} • %d{dd MMM yyyy HH:mm:ss,SSS} • and so on. PolicyCenter uses %d{ISO8601} by default.
%F	Name of the Java source file. Because the PolicyCenter logging API is a wrapper around log4j, %F returns a file name for the PolicyCenter logging API. If you want file names in your log messages, include them specifically in the message rather than by using %F in the conversion pattern.
%l	Abbreviated format for %F%L%C%M. This outputs the Java source file name, line number, class name and method name. Because the PolicyCenter logging API is a wrapper around log4j, the information returned is for the PolicyCenter logging API. If you want information such as class and method names in your log messages, include them specifically in the message rather than by using %l in the conversion pattern.
%L	Line number in Java source. Because the PolicyCenter logging API is a wrapper around log4j, %L returns a line number from the PolicyCenter logging API. If you want line numbers in your log messages, include them specifically in the message rather than by using %L in the conversion pattern.
%m	The log message.
%M	Name of the Java method. Because the PolicyCenter logging API is a wrapper around log4j, %M returns info string. If you want method names in your log messages, include them specifically in the message rather than by using %M in the conversion pattern.
%n	Newline character of the operating system. This is preferable to entering \n or \r\n as it works across platforms.
%p	Priority of the message. Typically, either FATAL, ERROR, WARN, INFO or DEBUG. You can also create custom priorities in your own code.
%r	Number of milliseconds since the program started running.
%t	Name of the current thread.
%throwable	Include a throwable logged with the message. Available format is: • %throwable – Display the whole stack trace. • %throwable{n} – Limit display of stack trace to n lines. • %throwable{none} – Equivalent of %throwable{0}. No stack trace. • %throwable{short} – Equivalent of %throwable{1}. Only first line of stack trace.
%X	The nested diagnostic context. You can use this to include server and user information in logging messages. Specify a key in the following format to retrieve that information from the nested diagnostic context: %X{*key*}. The following keys are available: • server • user • userID • userName For example, to include the server name, add %X{server}. For example, to include the server name, add %X{server}. There are three options for logging user information in logging patterns: user – prints the numeric opaque ID for the user userID – a unique user ID string, such as "aapplegate" userName – a real name, such as "Andy Applegate" For any of these, specify the minimum and the maximum size of the field. For example: %-16.16X{userName}. If the actual value is shorter than the minimum field size, the user identifier gets padded with spaces on the right. If the actual value is longer than the maximum size of the field, the user identifier gets truncated from the left. The user key lists a sequence number assigned to the user by the server and is not very informative. To include user login ID information, instead use the userID key.

Formatting Log Messages

You can specify the format of information in log messages by using a conversion pattern for the characters listed previously. These conversion patterns are closely related to conversion patterns used by functions such as `printf()` in the C language. Add the format specification between the percent sign and the letter in the conversion pattern. The following table describes conversion patterns available with log4j.

Pattern	Description
`%N`	Specifies a minimum width of *N* for the output, where *N* is an integer. If the output is less than the minimum width, the logger pads the output with spaces. Text is right-justified. For example, to specify a minimum width of 30 characters for the logging category, add `%30c` to the conversion pattern.
`%-N`	Left-justifies the output within the minimum width of *N* characters, where *N* is an integer. For example, to have the logging category left justified within a minimum width of 30 characters, add `%-30c` to the conversion pattern. The default output is right-justified.
`%.N`	Specifies a maximum width of *N* for the output, where *N* is an integer. For example, to have the logging category output have a maximum width of 30 characters, add `%.30c` to the conversion pattern. The logger truncates output from the beginning if it exceeds the maximum width.
`%M.N`	The logger pads with spaces to the left if output is shorter than *M* characters. If output is longer than *N* characters, then the logger truncates from the beginning.
`%-M.N`	The logger pads with spaces to the right if output is shorter than *M* characters. If output is longer than *N* characters, then the logger truncates from the beginning.

Listing Logger Categories

The `logging.properties` file might or might not contain all available logging categories. This is because both internal PolicyCenter code or custom integration code can define logging categories. Use the following command from `PolicyCenter/admin/bin` to list available PolicyCenter logging categories:

```
system_tools -password password -loggercats
```

Or, you can use the `SystemToolsAPI` web service method `getLoggingCategories`.

The server must be running before you can issue the command or call the API successfully.

The `system_tools` command and the web service method only list logging categories defined by PolicyCenter. Some third-party components, such as JGroups, provide their own categories.

For information about logging for plugins, see “Logging” on page 613 in the *Integration Guide*.

The logging categories defined by PolicyCenter include:

Category	Description
`AddressAutoSync`	Logging for the synchronization of contact data between ContactManager and PolicyCenter.
`AddressCorrection`	Logging for the address correction performed by the Geodata implementation.
`Api`	Used for all SOAP API calls.
`Api.AccountAPI`	Used for calls to the SOAP `AccountAPI`.
`Api.AddressAPI`	Used for calls to the SOAP `AddressAPI`.
`Api.CancellationAPI`	Used for calls to the SOAP `CancellationAPI`.
`Api.ContactAPI`	Used for calls to the SOAP `ContactAPI`.
`Application`	Internal Guidewire base logger category for application logging.

Category	Description
`Application.Addressbook`	Guidewire platform code that interacts with ContactManager. NOTE: This is not the category used for ContactManager.
`Application.Addressbook.Config`	Logging for the configuration of the Address Book subsystem.
`Application.Financials`	Logging for the financials subsystem.
`Application.Forms`	Logging for forms.
`Application.JobProcess`	Logging for job processes.
`Application.Locations`	Logging for locations.
`Application.PolicyHolds`	Logging for policy holds.
`Application.Quoting`	Logging for quotes on jobs (policy transactions). The `Quoting` category logs at the `INFO` level.
`Application.Rating`	Logging for messages related to Guidewire Rating Management.
`Application.Rating.RateTableManagement`	Logging for messages related to rate table management.
`Application.Rating.Rateflow`	Logging for messages related to creating and executing rate routines.
`Assignment`	Base logger category for assignment logging.
`Availability`	Logging of the determination of availability of elements in PolicyCenter. Availability criteria for each element are based on the element type. For example, one criterion could be the effective date of an element. If the effective date is not prior to or equal to today's date, the element would not be available.
`Configuration`	Logging for configuration problems in such areas as in security configuration, PCF configuration, locale configuration and so forth.
`Configuration.geodata`	Logging for configuration issues related to the Geodata daemon.
`Configuration.ProductModel`	Logging for configuration issues related to the product model.
`Datagen`	Internal Guidewire logger category for testing.
`GeneralExecution`	The default logger category for logging done by the `gw.api.util.Logger` class.
`Geodata`	Base logging category for the Geodata daemon.
`Globalization.PhoneNormalizer`	Logging for the Phone Number Normalizer process.
`Import`	Logging for import of XML data into PolicyCenter.
`Messaging`	Base logging category for the messaging system.
`Messaging.Email`	Logging of email message destination and email generation code.
`Messaging.Events`	Logging of events. For more information about events and messaging, see "Messaging and Events" on page 289 in the *Integration Guide*.
`Plugin`	Logging for all calls into any plugin. This is also the "parent category" for individual plugin categories. For more information on logging with plugins, see "Logging" on page 613 in the *Integration Guide*.
Plugin.ContactSystemPlugin	Logging for the `ContactSystemPlugin`. See "Contact Integration" on page 523 in the *Integration Guide*.
`Plugin.Document`	Logging for the document plugin.
`Plugin.IAccountPlugin`	Logging for the account plugin, which PolicyCenter uses to configure how PolicyCenter manages accounts. For more information about the account plugin, see "Account Plugin Details" on page 151 in the *Integration Guide*.
`Plugin.IAddressBookAdapter`	**Deprecated**. Logging for the `IAddressBookAdapter` address book plugin, which PolicyCenter uses to communicate with ContactManager. For more information on `IAddressBookAdapter`, see "Contact Integration" on page 523 in the *Integration Guide*.

Category	Description
`Plugin.IApprovalAdapter`	Logging for the approval plugin, which PolicyCenter uses to determine whether items need approval and to whom PolicyCenter assigns the item if it does need approval.
`Plugin.IBillingRetrievalPlugin`	Logging for the billing retrieval plugin, which PolicyCenter uses to retrieve billing info from an external system. For more information, see "Billing Integration" on page 453 in the *Integration Guide*.
`Plugin.IBillingSummaryPlugin`	Logging for the `IBillingSummary` plugin. See "Implementing the Billing Summary Plugin" on page 497 in the *Integration Guide*.
`Plugin.IBillingSystemPlugin`	Logging for the `IBillingSystem` plugin. See "Implementing the Billing System Plugin" on page 487 in the *Integration Guide*.
`Plugin.IClaimSearchPlugin`	Logging for the claim search plugin, which PolicyCenter uses to search for claims against a policy in an external system. For more information, see "Claim Search from PolicyCenter" on page 505 in the *Integration Guide*.
`Plugin.IContactSearchAdapter`	Logging for the `IContactSearchAdapter` plugin. For more information on `IContactSearchAdapter`, see "Contact Integration" on page 523 in the *Integration Guide*.
`Plugin.ICustomPickerAdapter`	Logging for the custom picker plugin. This plugin is deprecated and will be removed in a future release.
`Plugin.IDocumentMetadataSourceBase`	Logging for the documentation management metadata plugin. For more information on documentation management plugins, see "Document Management" on page 191 in the *Integration Guide*.
`Plugin.IJobNumberGenPlugin`	Logging for the job number generator plugin, which PolicyCenter uses to generate job numbers in place of using the built-in algorithm. For more information, see "Job Number Generator Plugin" on page 152 in the *Integration Guide*.
`Plugin.ILossHistoryPlugin`	Logging for the loss history plugin, `ILossHistoryPlugin`, which retrieves the loss history for an account or policy from an external system. For more information about `ILossHistoryPlugin`, see "Loss History Plugin" on page 157 in the *Integration Guide*.
`Plugin.IPolicyNumGenPlugin`	Logging for the policy number generator plugin, `IPolicyNumGenPlugin`, which generates policy numbers. For more information about `IPolicyNumGenPlugin`, see "Policy Number Generator Plugin" on page 150 in the *Integration Guide*.
`Plugin.IReinsuranceCedingPlugin`	Logging for the `IReinsuranceCeding` plugin. See "Reinsurance Ceding Plugin" on page 414 in the *Integration Guide*.
`Plugin.IReinsuranceConfigPlugin`	Logging for the `IReinsuranceConfig` plugin. See "Reinsurance Configuration Plugin" on page 411 in the *Integration Guide*.
`Plugin.IReinsurancePlugin`	Logging for the `IReinsurance` plugin. See "Reinsurance Plugin" on page 405 in the *Integration Guide*.
`Plugin.ITerritoryCodePlugin`	Logging for the territory code plugin, `ITerritoryCodePlugin`, which you can use to retrieve `Territory` entities from an external system. For more information about `ITerritoryCodePlugin`, see "Territory Code Plugin" on page 251 in the *Integration Guide*.
`Plugin.IVinPlugin`	Logging for the `IVinPlugin`, which returns vehicle information from a standard vehicle identification number. For more information about `IVinPlugin`, see "Vehicle Identification Number Plugin" on page 253 in the *Integration Guide*.
`Profiler`	Used to log the Guidewire built-in profiler.
`RuleEngine`	Do not use. Use `RuleExecution` instead.
`Rules`	Do not use. Use `RuleExecution` instead.
`Security`	Internal Guidewire base logger category for security logging.
`Server`	Internal Guidewire base logger category for server/platform logging.

Category	Description
Server.Admin Export/Import	Logging for the process of importing and exporting of administrative data.
Server.Archiving	Logging for messages generated during both graph construction and validation and by workers in the archive work queue.
Server.Archiving.DocumentUpgrade	Messages generated during upgrade of archived XML.
Server.Archiving.Success	Logging for messages generated for each entity that is successfully archived.
Server.BatchProcess	Batch process logging. Starting, stopping, and so forth. You can not configure a particular batch process to use a different logger.
Server.BatchProcess.PCStatisticsCalculator	Logging for the PCStatisticsCalculator batch process.
Server.BatchProcess.PendingAccountDataUpdate	Logging for the PendingAccountDataUpdate batch process.
Server.Cluster	Logging for PolicyCenter messages related to clustering. The clustering implementation, JGroups, has its own logging category, org.jgroups. The default logging.properties file includes a section for this category.
Server.ConcurrentDataChangeException	Logging for concurrent data changes, in which two commits to the database alter the same object.
Server.Database	Used for all database type logging, transactions and so forth.
Server.Database.Staging	Used for staging table related logging.
Server.Database.Upgrade	Messages about the database upgrade.
Server.Global Cache	Logs messages related to what is happening in the cache. PolicyCenter only uses this category to print warnings and errors.
Server.Preload	Provides DEBUG level logging of all actions during server preloading of Gosu classes. See “Preloading Gosu Classes” on page 98 in the *Configuration Guide*.
Server.Profiler	Logging for the Guidewire Profiler.
Server.RunLevel	Used by “startables” during their start and stop methods and to log run level changes.
Server.Workflow	The base logger category for all workflow engine logging.
Server.Workflow.WorkflowResourceController	Logging of errors on failure to parse a workflow definition file.
Studio	Internal Guidewire base logger category for Guidewire Studio logging.
Test	Guidewire internal category.
Test.Database	Guidewire internal category.
UserInterface	Internal Guidewire base logger category for user interface logging.
UserInterface.Performance	Logging for events such as frame rendering and widget events. To log user interface performance messages, set the logging level for the UserInterface.Performance category to TRACE. All events contain a frame reference. Messages regarding frame rendering are tagged with the text PERFLOG-RENDER. Messages about widget events are tagged with the text PERFLOG-WIDGET. The timestamp and user ID are available from the context. You can configure the UserInterface.Performance logging category to list the user ID and timestamp by including the fields %X{userID} and %d in the layout conversion pattern in logging.properties. PCF element information is available within the widget event messages.
UserInterface.JobWizard	Logging for the PolicyCenter Job wizard.
Workqueue	Base logging category for work queue functionality. For more information on work queues, see “Batch Processing” on page 99.
Workqueue.Instrumented	Trace logging of commits to the instrumented worker table.

Category	Description
`Workqueue.Item`	Logging for processing of work queue work items.
`Workqueue.Runner`	Logging of informational messages for a worker, including time it starts, stops and suspends, various errors while running, and a trace level for each execution.

In some cases, a Guidewire product might have not have a particular plugin. In those cases, the category exists but PolicyCenter does not use the category. Along with the default logging categories, each application has its own unique categories.

Making Dynamic Logging Changes without Redeploying

You can make dynamic changes to the logging configuration without having to redeploy PolicyCenter.

Changing the database logging level dynamically does not make any difference to existing database connections. The new logging level only take effects for new database connections. For example, if the database log level is set to debug, PolicyCenter logs all SQL statements. However if the debug logging level is set dynamically, PolicyCenter only logs SQL statements for new connections created within the connection pool. If an existing connection is used, dynamically changing the logging level to debug will not have any affect. To make sure that the logging level is set consistently for all database connections, set the log level in `logging.properties` and restart the server.

Reloading the Logging Configuration

You can make an immediate logging change in the PolicyCenter server without having to first redeploy PolicyCenter. Update the `logging.properties` file with your changes. Then, update the logging configuration on the server by using either the `system_tools` command-line utility or `SystemToolsAPI` web service.

To update the logging configuration from the command line, use the following command from the `admin/bin` directory:

```
system_tools -password gw -reloadloggingconfig
```

To update the logging configuration from the web service, call the following method:

```
SystemToolsAPI.reloadLoggingConfig()
```

Either approach reloads the logging configuration from the `logging.properties` file.

Temporarily Changing a Logging Level

You can temporarily change the logging level for a logger by using either the `system_tools` command-line utility, `SystemToolsAPI` web service, or the **Set Log Level** info page.

To update a logging level from the command line, use the following command from the `admin/bin` directory:

```
system_tools -updatelogginglevel logger level
```

To update the logging configuration from the web service API, call the following method:

```
SystemToolsAPI.updatelogginglevel(logger,level)
```

Either approach sets the logging level of the logger. Supply a `String` representation of either the logger category for category-based logging or the Java class name for class-based logging. For the root logger, specify `RootLogger` for the logger name. The change in the logging level persists only while the PolicyCenter server is running.

You can also set the logging level with the **Set Log Level** screen in PolicyCenter. See “Set Log Level” on page 143.

Logging levels that you change dynamically remain in effect only while the server is running. If you restart the server, then PolicyCenter resets logging behavior to what you specified in the `logging.properties` file.

chapter 3

Configuring and Maintaining the PolicyCenter Database

This topic discusses key issues for configuring and maintaining the PolicyCenter database. While PolicyCenter automatically handles most changes to its schema, involve a database administrator in tuning and managing the database server.

This topic includes:

See also

- "Configuring the Database" on page 27 in the *Installation Guide*
- "Configuring a Database Connection" on page 66 in the *Installation Guide*
- See the *Guidewire Platform Support Matrix* for current system and patch level requirements. The *Guidewire Platform Support Matrix* is available from the Guidewire Resource Portal at `https://guidewire.custhelp.com/app/resources/products/platform`.

Database Best Practices

Guidewire recommends the following best practices for the database:

- If you need to perform any database maintenance tasks, such as applying a patch, shut down application servers running PolicyCenter that are attached to the database. Restart the application servers after the database maintenance is complete.
- For Oracle databases, keep the Oracle default settings as much as possible. For example, do not set a 4K blocksize. Consult with Guidewire if you want to change the default Oracle settings.
- Do not insert directly into tables managed by PolicyCenter. This can cause the data distribution tool to fail and cause other problems. See “Guidewire Database Direct Update Policy” on page 36.
- Do not add lots of `mediumtext` and `CLOB` columns to a table.
- Do not add an index outside of PolicyCenter and not declare it in an extension file.
- Monitor how storage performs. If I/O (input/output) times are slower than 10ms, something is wrong.
- Monitor tablespace size allocations and disk space, so PolicyCenter does not run out of space.
- Back up the database periodically to support disaster recovery options. See “Backing up the PolicyCenter Database” on page 38.
- Update database statistics periodically so that the query optimizer selects an efficient plan for executing application queries. See “Configuring Database Statistics” on page 42.
- Run consistency checks on the database, especially after importing data. See “Checking Database Consistency” on page 40.

Guidewire Database Direct Update Policy

PolicyCenter runs on SQL-based Relational Database Management Systems (RDBMS). You can use SQL or other query tools directly in a read-only manner to extract or view data. Note that such read-only queries, depending on their scope and how they are written, can negatively effect overall database performance even though they do not modify any data. Guidewire recommends that you run SQL queries in a replica or copy of their production database, rather than the production database itself. For applications such as data warehouses and intensive reporting, Guidewire recommends that you explore mechanisms for replicating or summarizing data into a production reporting database for this purpose. This practice can help unexpected production performance issues due to intensive reporting requirements or lengthy queries.

Do not use SQL queries or similar direct-to-database update tools to add or modify any data associated with PolicyCenter. The internal application logic, embedded in PolicyCenter application code and APIs, maintains a variety of data and metadata that is related to your application data. This might not be obvious from review of the RDBMS table structure. Examples include:

- calculations of summary table data for reporting.
- caching of application data in memory for faster access.
- tracking of state information associated with the underlying data, such as the processing state of an integration message.

For this reason, never use SQL to directly update the underlying RDBMS. Any such direct SQL updates could leave the data in an inconsistent state. Guidewire might require you to restore the database to a previous state if you require support after performing such an update query. Guidewire Support will not be able to assist you with diagnosing and correcting application issues caused by your database queries. It would be your responsibility to restore PolicyCenter to a consistent state.

If you have a legitimate need to update underlying application data, Guidewire recommends that you use Guidewire APIs, either in Java or Gosu, to perform the necessary updates. This ensures that no critical side effects of the updates are missed in the process of altering the data. Using Guidewire APIs to update application data is safer than using SQL queries with regard to consistency. However, with any programming language or API it is still possible to update data incorrectly or in ways that do not perform well. Therefore, when using the APIs, Guidewire strongly recommends that you review your intended updates with your Guidewire Support Partner and/or Guidewire Professional Services team.

In rare cases, for example, in situations in which no API existed to correct a data corruption problem, Guidewire might advise customers on using SQL queries to correct these problems. In these cases, the SQL queries used to update the database must be written by or approved by Guidewire. This is to ensure that the correct logic is used and that all potential side effects are taken into account. Do not apply any other SQL queries to modify data in a PolicyCenter database. Guidewire will not review or provide such queries for situations where an API or supported alternate method is available.

Configuring Connection Pool Parameters

If you experience slow performance, it could be that the PolicyCenter server is not allocating enough database connections. If all database connections are in use, any client attempting to connect to the server must wait until a connection is free. By default, PolicyCenter periodically tests connections in the connection pool and evicts idle connections and those that fail with an exception when tested with a simple query. You can configure this behavior and set other connection pool parameters by modifying or adding attributes to the `<dbcp-connection-pool>` element within the `<database>` element of `database-config.xml`. Access this file from the Guidewire Studio **Project** window under **configuration** → **config**.

Attribute	Description
`max-active`	The maximum number of active connections. A reasonable initial value for this is about 25% of the number of users that you expect to use PolicyCenter at the same time. When `max-active` is reached, the pool is said to be exhausted. If `max-active` is set to a negative number, then there is no limit. The default value is `-1`.
`max-idle`	The maximum number of objects that can sit idle in the pool at any time. If `max-idle` is set to a negative number, then there is no limit. The default value is `-1`.
`max-wait`	The maximum time in milliseconds that the data source waits for a connection before one becomes available in the pool to service. The default is `30000`.
`min-evictable-idle-time`	The time, in milliseconds, that an object may sit idle in the pool before it is eligible for eviction due to idle time. When non-positive, no object will be dropped from the pool due to idle time alone. This setting has no effect unless `time-between-eviction-runs` is greater than `0`. The default is `300000`.
`num-tests-per-eviction-run`	The number of idle connections PolicyCenter tests each eviction run. The default is `3`.
`time-between-eviction-runs`	The time, in milliseconds, that PolicyCenter waits between eviction runs. The default is `60000`.
`test-on-borrow`	Whether PolicyCenter tests a connection by running a simple query as PolicyCenter first borrows the connection from the connection pool. The default is `false`.
`test-on-return`	Whether PolicyCenter tests a connection by running a simple query as PolicyCenter returns the connection to the connection pool. The default is `false`. Since PolicyCenter returns connections used for just a query to the pool immediately after the query, running a test query on return to the pool could affect performance.

test-while-idle	Whether PolicyCenter performs eviction runs on the connection pool. During an eviction run, PolicyCenter scans the connection pool and tests a number of idle connections equal to numTestsPerEvictionRun. If a connection has been idle more than min-evictable-idle-time milliseconds, PolicyCenter evicts the connection from the pool. Otherwise, PolicyCenter executes a simple query on the connection. If the query fails with an exception, then PolicyCenter evicts the connection. The default value of test-while-idle is true.
when-exhausted-action	Specifies the behavior of the borrowObject() method when the pool is exhausted. If when-exhausted-action equals fail, borrowObject() throws a NoSuchElementException. If when-exhausted-action equals grow, borrowObject() creates a new object and returns it, essentially making max-active meaningless. If when-exhausted-action equals block, borrowObject() blocks (invokes Object.wait()) until a new or idle object is available. If a positive max-wait value is supplied, then borrowObject() blocks for at most that many milliseconds, after which a NoSuchElementException will be thrown. If max-wait is not positive, the borrowObject() method blocks indefinitely.

You can view, but not edit, many of these parameters from within PolicyCenter at **Server Tools** → **Info Pages** → **Database Parameters** → **Database Connection Pool Settings**.

The parameters listed previously apply if you are using the default connection pool. If you instead use the application server connection pool, these settings do not apply. Configure the application server connection pool through the administration console provided with the application server. See “Configuring PolicyCenter to Use a JNDI Data Source” on page 72 in the *Installation Guide*.

Backing up the PolicyCenter Database

PolicyCenter stores most information in the database, so the most important part of backing up the system is taking frequent database backups. Consult the documentation for your database management system for tools and techniques to backing up the database.

You can perform a hot (also called dynamic) or cold backup of the PolicyCenter database. You can take a hot backup while users are still accessing PolicyCenter. Read your database documentation to understand the risks involved in hot backups. If you plan on taking a cold backup, you must put the application server in maintenance mode before performing the backup.

After restoring a PolicyCenter database from a backup, instruct PolicyCenter to rebuild its database statistics. See “Configuring Database Statistics” on page 42.

In addition to backing up the database, maintain a backup of the PolicyCenter configuration. This is especially important for any files that you modify as part of installation or configuration of PolicyCenter. All of these files are located within the PolicyCenter/modules/configuration/config directory, making it easy to keep those files in a source control system, if desired.

In the case of a complete system failure, with proper backups you can reinstall the PolicyCenter WAR or EAR file on a new server, connecting to the same database. In the case of a database failure, you would be able to restart PolicyCenter from the last database backup.

Understanding and Authorizing Data Model Updates

When you start PolicyCenter, it compares system metadata (the description of the objects and tables in the `config` directory) to the database to see if they match. For example, if a new extension has been added since the last server start, the database and metadata do not match. If these two do not match, PolicyCenter attempts to update the database to match the metadata. This type of update to the data model is different than a product version upgrade, which includes more extensive changes to the database.

If the `autoupgrade` attribute of the `database` connection settings block in `database-config.xml` is set to `true`, then, at startup, PolicyCenter makes database changes without waiting for confirmation. If `autoupgrade` is `false`, PolicyCenter reports the need to update the database to the console and sets the run level to `shutdown`.

Disable automatic updates in production environments to prevent unexpected changes. To disable automatic updates, set `autoupgrade` to `false`. During configuration and testing, it is more convenient to have database changes execute automatically. In this case, set `autoupgrade` to `true`. In either case, the first time you start PolicyCenter, it must perform a database update.

You can trigger a database update without a change in the metadata by increasing the version number in `config/extensions/extensions.properties`.

The update process calculates current checksums for all the XML files in the data model. It then compares them with historical checksums stored in the `SystemParameter` entity. If the values differ, then PolicyCenter updates the database to match the metadata. As the last step in the update, PolicyCenter updates the `SystemParameter` entity with the current checksums.

If during the update PolicyCenter creates a new table, then it also generates a unique index for the table. The `TableRegistry` entity stores this information. In this way, PolicyCenter guarantees uniqueness.

Before completing, PolicyCenter again verifies the data model against the physical database. You can run the schema verifier from the command line with the following command:

```
system_tools -password password -verifydbschema
```

If, for some reason, the model and database disagree, PolicyCenter writes warnings to the log and, if possible, suggests corrective actions. Take the corrective action if prompted to do so.

The database update takes a number of actions that can impact database statistics. The upgrade might recreate tables, drop indexes and create new indexes. The update process writes an `updatedatabasestatistics.sql` file to the `work` directory on the application server. The contents of the file are dependent on the SQL executed during the update and the statistics parameters of the `<database>` element. After an update, run the `updatedatabasestatistics.sql` script to update database statistics. See “Configuring Database Statistics” on page 42 for more information about setting the statistics parameters.

> **Note:** If the server is interrupted during a database update for any reason, the server resumes the update upon restart. PolicyCenter accomplishes this by storing the steps in the database and marking them completed as part of the same database transaction that applies a change. This only applies to data model updates and does not apply to product version upgrades.

For more details on configuration changes that require database updates, see “Modifying the Base Data Model” on page 211 in the *Configuration Guide*.

PolicyCenter includes an **Upgrade Info** page that provides detailed information about the database upgrade. The **Upgrade Info** page includes information on the following:

- version numbers before and after the database upgrade
- configuration parameters used during the database upgrade
- SQL queries for version checks that test if the database is in condition to be upgraded
- changes made to specific tables, including which version triggers modified the table or its data and the SQL statement executed to make each change

- version triggers that the upgrade ran, including which tables the trigger ran against, a description, the SQL statement run against each table and the start and end time
- a list of upgrade steps, including the table on which the step operated
- a table registry including table IDs before and after upgrade

The database upgrade deletes upgrade instrumentation information for prior database upgrades. If the database upgrade detects any prior upgrade instrumentation data, it reports a warning and deletes the data. If you have run previous database upgrades, and you want to preserve upgrade instrumentation details, download this information.

To download upgrade instrumentation details

1. Start the PolicyCenter server if it is not already running.
2. Log in to PolicyCenter with the superuser account.
3. Press `ALT+SHIFT+T` to access **System Tools**.
4. Click **Info Pages**.
5. Select **Upgrade Info** from the **Info Pages** drop-down.
6. Click **Download** to download a ZIP file containing the detailed upgrade information.

Checking Database Consistency

PolicyCenter includes a number of database consistency checks. These checks determine if any unusual conditions exist in the PolicyCenter database such as orphaned child records or inconsistencies between properties. Problems reported by the checks are sent to the console and logged in the `pclog.log` file. This mode is especially useful during trial periods or for testing converted data.

Check the database consistency on a regular basis to identify potential problems. Run consistency checks before and after a database upgrade and before importing data into a production database.

Running Consistency Checks from PolicyCenter

Guidewire recommends that you view and run consistency checks from the **Consistency Checks** page in PolicyCenter. The **Consistency Checks** page lists which consistency checks are available for specific tables. You can also use this page to view the results of running consistency checks previously. If there are consistency check errors, the results include SQL queries that you can use to identify records that violated the consistency check.

See “Consistency Checks” on page 147 for more information.

Running Consistency Checks with System Tools

It is also possible to launch database consistency checks from the command line. Launching consistency checks from the command line is primarily useful for scheduling checks to run on a regular basis. Use the following command:

```
system_tools -checkdbconsistency -password password
```

The `system_tools` utility is located in the `PolicyCenter/admin/bin` folder. See “System Tools Command” on page 162 for more information.

This tool has two optional arguments:

```
-checkdbconsistency tableSelection checkTypeSelection
```

The *`tableSelection`* argument can be specified as:

- `all` – Run consistency checks on all tables.

- *table name* – The name of a single table on which to run checks.
- tg.*table group name* – The name of a table group. Table groups are defined in the <database> element of database-config.xml. For more information, see "Defining Table Groups" on page 69 in the *Installation Guide*.
- @*file name* – A file name with one or more valid table names or table group names entered in comma-separated values (CSV) format. Prefix table group names with tg., such as tg.MyTableGroup. You can combine table groups and individual table names in the same file.

The *checkTypeSelection* can be specified as:

- all – Run all consistency checks on the specified tables.
- *check name* – The typecode value of a single consistency check to run.
- @*file name* – A file name with one or more valid consistency check names entered in comma-separated values (CSV) format.

If you specify one optional argument, you must specify both.

The -checkdbconsistency option runs consistency checks as an asynchronous batch process.

Using a larger number of threads can help performance as long as your server can process the threads. Guidewire recommends starting with five threads. If too many threads are used, there is a greater chance that current users experience reduced performance if the database server is fully loaded. You can adjust the number of threads by modifying the number of worker instances used by the consistency check work queue.

Use the typecode value to specify a consistency check type as an argument or in a file. To see which consistency check types are available for each table, search for the table on the **View consistency checks definitions** tab of the **Info Pages** → **Consistency Checks** page. This page lists the consistency checks available by name. Then select the **Run consistency checks** tab. For **Check all types?**, select **Specify types**. Use the **Type Code** value to specify the consistency checks that you want to run. The Typelists section of the *PolicyCenter Data Dictionary* also lists available consistency check types. The typelist is ConsistencyCheckType.

Results of the consistency checks are available from the **Consistency Checks** page. See "Consistency Checks" on page 147.

The PolicyCenter log lists a check type for each consistency check that runs. Each check type in the log can have a value of either 0 or 1. A value of 0 indicates that PolicyCenter expects the check to return zero results if the database is consistent. A value of 1 indicates that PolicyCenter expects the check to have a different return value. PolicyCenter uses the check type for custom checks. If a check type 0 fails, the log records a failure description that PolicyCenter expected the query issued by the check to return zero rows. The log includes the SQL query run by the check and the number of inconsistencies returned by the query. If a check type 1 fails, the log includes a specific failure description.

> **Note:** The check type in the PolicyCenter log is not the same as the check type shown in the Typelists section of the *PolicyCenter Data Dictionary.*

Running consistency checks using the -checkdbconsistency option can take a long time. If the connection times out while running this command, try the following:

- Run consistency checks on fewer tables at a time by using the arguments shown previously.
- Increase the number of worker instance threads used by the consistency check work queue. See "Configuring Number of Threads for Consistency Checks" on page 42.

Running Consistency Checks when the Server Starts

For development environments with very small data sets, you can enable consistency checks to run each time the PolicyCenter server starts. To do this, set the `checker` attribute of the `database` block to `true` in `database-config.xml`. By default, this option is set to `false`.

IMPORTANT Running these consistency tests when starting the server can take a long time, impact performance severely, and possibly time out on large datasets. Set `checker` to `false` under most circumstances. Guidewire recommends that you do not set `checker` to `true` except in development environments with very small test data sets.

If the database connection times out when running consistency checks by using the `checker` attribute, increase the number of threads by increasing the value of `ConsistencyCheckerThreads` in `config.xml`.

Configuring Number of Threads for Consistency Checks

Using a larger number of threads for consistency checks can help performance as long as your server can process the threads. Guidewire recommends starting with five threads. If too many threads are used, there is a greater chance that current users experience reduced performance if the database server is fully loaded.

PolicyCenter uses the work queue mechanism to run consistency checks. You can therefore configure work queue and worker attributes for consistency checks in the `work-queue.xml` file. Access this file in Guidewire Studio at **configuration** → **config** → **workqueue**. Set parameters for the work queue with `workQueueClass` set to `com.guidewire.pl.system.database.checker.DBConsistencyCheckWorkQueue`. For example:

```
<work-queue workQueueClass="com.guidewire.pl.system.database.checker.DBConsistencyCheckWorkQueue">
   <worker instances="5" batchsize="10"/>
</work-queue>
```

In this example, the consistency check work queue is configured to use five worker threads and for each worker to check out ten work items at a time.

After you edit `work-queue.xml`, you must rebuild and redeploy PolicyCenter.

For more information about work queue configuration, see “Configuring Work Queues” on page 105.

For development environments with very small data sets, you can run consistency checks on server startup by setting `checker` to `true` in the `database` block of `database-config.xml`. For this configuration you can modify the number of threads by setting the `value` of `ConsistencyCheckerThreads` in `config.xml`.

Configuring Database Statistics

Database statistics are metadata that describe the underlying database. For example, database statistics store row counts in a table, how the data is distributed in a table, and much more. A database management system uses statistics to determine query plans to optimize performance.

PolicyCenter provides database statistics generation designed specifically for how the PolicyCenter application and data model interact with the physical database.

With Oracle, generating database statistics from the database management system can potentially create statistics that cause PolicyCenter to select a bad plan for execution of SQL queries. Therefore, always use PolicyCenter to generate database statistics, rather than by using the statistics generation provided with Oracle.

Guidewire recommends that Oracle implementations only update database statistics during quiet periods, such as weekends, so that these updates do not occur when BillingCenter is under heavy load. By default, updating statistics on a table or index invalidates existing query plans related to that table or index.

Guidewire further recommends that Oracle implementations use the `NO_INVALIDATE => AUTO_INVALIDATE` option when updating statistics. This is the default option. This option is also what the Guidewire Database Statistics batch process uses, unless the configuration parameter `DiscardQueryPlansDuringStatsUpdateBatch` is set to `true`. Setting `NO_INVALIDATE => FALSE` to force immediate invalidation of query plans has a high likelihood of causing issues with concurrent batch updates. Using `AUTO_INVALIDATE` greatly reduces this risk. Ideally, set the `_optimizer_invalidation_period` parameter to a low value (a few minutes) to reduce the time window during which Oracle might invalidate a plan.

For SQL Server, have SQL Server update database statistics. Do not use PolicyCenter to update the statistics or run the statistics statements that PolicyCenter can create. Guidewire requires that you set `Auto Create Statistics` and `Auto Update Statistics` to `true` on the database account used for PolicyCenter.

Updating database statistics can take a long time on a large database. Only collect statistics if there are significant changes to data, such as after a major upgrade, after using the `zone_import` command, or if there are performance problems. Under normal operating conditions, you do not need to update database statistics on the PolicyCenter server often. If you encounter performance problems or degradation related to the database, check the **Database Statistics** page on the **Info Pages** section of the **Server Tools**. If the page shows suspicious or inaccurate statistics, update database statistics. If the data change is high, consider using a weekly or biweekly schedule for updating statistics.

The database statistics batch process is resource-intensive. To prevent application administrators from accidentally running the process, it can only be started from the command line. Consult with your Database Administrator before starting the database statistics process.

You can also run a process to only update statistics for tables that have had a configurable percentage of data changed since the last statistics process was run.

PolicyCenter automatically updates specific database statistics during an upgrade, in conjunction with selected batch processes, or during the `zone_import` process.

For database upgrades, PolicyCenter updates database statistics for objects that the upgrade process changes significantly. For optimum performance, generate incremental database statistics after performing an upgrade between major versions.

Some PolicyCenter batch processes use work or scratch tables to store intermediate calculations. Other batch processes populate denormalized tables that PolicyCenter uses internally for performance reasons. These processes can update database statistics on the scratch tables and denormalized tables during their execution.

Using the **Database Statistics** page, you can see if any statistics are out of date. See “Database Statistics” on page 149. Guidewire recommends that you archive database statistics as standard practice. This ensures that you have a record of the database history that can be reviewed if necessary.

IMPORTANT Have your database administrator (DBA) review these statistics with you.

Commands for Updating Database Statistics

You can use command line commands to explicitly update database statistics or to generate the SQL statements to update statistics. The commands are:

Type	Command

Full
Generates database statistics for every table in the PolicyCenter database.
	To update statistics for all tables:
	`system_tools -password password -updatestatistics description false`
	To generate database statistic SQL statements for all tables:
	`system_tools -password password -getdbstatisticsstatements`
	You can use the results purely as a reference, or you can edit the statements and execute them outside of PolicyCenter. Statements are grouped by table.

Incremental
Generates database statistics for tables where the change in the table data caused by inserts and deletes exceeds a certain percentage threshold. The threshold is specified by the `incrementalupdatethresholdpercent` attribute on the `<databasestatistics>` element. The default is 10 percent.
	To update statistics for tables exceeding the change threshold:
	`system_tools -password password -updatestatistics description true`
	The change threshold is defined by the `incrementalupdatethresholdpercent` attribute of the `databasestatistics` element in `database-config.xml`.
	This process does not update statistics on any table that has locked statistics.
	To generate database statistic SQL statements for tables exceeding the change threshold:
	`system_tools -password password -getincrementaldbstatisticsstatements`
	You can use the results purely as a reference, or you can edit the statements and execute them outside of PolicyCenter. Statements are grouped by table.

Configuring Database Statistics Generation

You can control which database statistics statements PolicyCenter generates by configuring the database connection in the `database-config.xml` file.

The `<databasestatistics>` element has the following format:

```
<databasestatistics samplingpercentage="integer" databasedegree="integer"
 incrementalupdatethresholdpercent="integer">
    <tablestatistics name="string" samplingpercentage="integer" databasedegree="integer"
     action="delete|keep|update">
    <indexstatistics name="string" databasedegree="integer" samplingpercentage="integer">
      <keycolumn name="string" keyposition="integer">
      <keycolumn name="string" keyposition="integer">
      ...
    <indexstatistics/>
    <histogramstatistics
     name="string"
     numbuckets="integer"
     databasedegree="integer"
     samplingpercentage="integer"/>
    ...
  </tablestatistics>
  ...
</databasestatistics>
```

In the following example, an Oracle database connection shows the use of these parameters:

```
<databasestatistics samplingpercentage="0" databasedegree="4"
 incrementalupdatethresholdpercent="15">
  <tablestatistics name="pc_table1" samplingpercentage="100" databasedegree="4">
    <histogramstatistics name="scheduledsenddate" numbuckets="254"/>
    <indexstatistics samplingpercentage="50">
      <keycolumn name="publicid" keyposition="1"/>
      <keycolumn name="retired" keyposition="2"/>
    </indexstatistics>
  </tablestatistics>
  <tablestatistics name="pc_table2" action="delete" />
</databasestatistics>
```

The above example configures the following database statistic generation behavior:

- Collect statistics on all PolicyCenter tables using the automatic sampling size and with a degree of parallelism of 4.
- The dbms_stats command for table pc_table1 samples 100% and uses a parallel degree of 4.
- The configuration defines a histogram with 254 buckets on cc_check.scheduledsenddate.
- The index on pc_table1 (publicID, retired) is sampled at 50% and statistics for the index are gathered.
- PolicyCenter deletes statistics on pc_table2 due to the attribute action="delete".

For Oracle, if you are using databasedegree greater than 1, it might be useful to set parallel_execution_message_size to 16384 in the server parameter file or init parameter file.

<databasestatistics>

This element specifies database statistic parameters that override the database defaults specified on the database. This element has the following attributes.

databasedegree	On Oracle, this attribute controls the degree of parallelism for each individual statement. The default is 1. PolicyCenter uses the value of this attribute for all statements. SQL Server ignores the databasedegree attribute.
samplingpercentage	On Oracle, this attribute controls the value of the estimate_percent parameter in the dbms_stats.gather_table_stats() SQL statements. You can set samplingpercentage to either an integer from 1 to 100 to directly set the estimate_percent value, or set samplingpercentage to 0 to set estimate_percent to AUTO_SAMPLE_SIZE. The default value is 0. PolicyCenter uses the database default for dbms_stats.gather_index_stats() statements. On SQL Server, the samplingpercentage attributes controls the value of the WITH FULLSCAN/SAMPLE PERCENT clause in the UPDATE STATISTICS statements. A value of 100, the default, translates into WITH FULLSCAN, as does a value of 0.
incrementalupdatethresholdpercent	Specifies the percentage of table data that must have changed since the last statistics process for the incremental statistics generation batch process to update statistics for the table.
numappserverthreads	On both Oracle and SQL Server, the numappserverthreads attribute controls the number of threads that are used to update database statistics for staging tables during import only. This import is launched using the table_import command. See "Table Import Command" on page 166. The value defaults to 1. If the value is greater than 1, then the application server assigns a table at a time to each thread as the thread becomes available. Each thread executes all of the database statistics statements for its assigned table. For all other statistics generation operations, set the number of threads by specifying the number of workers for the database statistics work queue. Set the instances attribute on the workers subelement of the work-queue element for the database statistics work queue. This element has workQueueClass="com.guidewire.pl.system.database.dbstatistics.DBStatisticsWorkItemWorkQueue".

The values you set for these attributes apply to all the tables in the database. You can fine tune these values and set specific values on individual tables by using the <tablestatistics> subelement. Setting values on a specific table overrides the values set on the database for just that table.

<tablestatistics>

You can use this element to override database-wide statistics settings defined on the <databasestatistics> element for a specific table. You can override the databasedegree (Oracle only), samplingpercentage, and statistic gathering behavior of PolicyCenter. Provide a name parameter to identify the table for which you want to set values:

```
<tablestatistics name="string" samplingpercentage="integer" databasedegree="integer"
 action="update|delete|keep"/>
```

By default, PolicyCenter on Oracle does not generate statistics on any table used for processing work items. PolicyCenter deletes any existing statistics on these tables whenever PolicyCenter updates statistics. You can override this behavior by using the action attribute of the <tablestatistics> element. You can set the action attribute to one of the following values:

update	Update the statistics on the table.
delete	Delete the statistics on the table. This value does nothing in SQL Server.
keep	Keep the existing statistics. PolicyCenter does not update statistics for any table where the user explicitly specifies keep as the value for the action attribute. This value affects any type of database.

The default value is update.

The <tablestatistics> element is optional. If you do not specify a <tablestatistics> element for a table, PolicyCenter uses the database-wide statistics defined on the <databasestatistics> element. If you do specify a <tablestatistics> element, the action attribute is required.

On Oracle, you can use the <indexstatistics> element or <histogramstatistics> subelements to override these values on specific indexes or histograms. SQL Server recognizes only the <histogramstatistics> elements.

indexstatistics

This is an optional element that overrides, for Oracle, the databasedegreee and samplingstatistics for an individual index. This element has no meaning in SQL Server.

The values you set override the database defaults for all dbms_stats.gather_index_stats statements on the named index. This element has the following format:

```
<indexstatistics name="string" databasedegree="integer" samplingpercentage="integer">
  <keycolumn name="string" keyposition="integer">
  <keycolumn name="string" keyposition="integer">
...
</indexstatistics>
```

You must specify a name attribute to identify the index. Then, you can specify a databasedegree attribute and/or a samplingpercentage attribute.

histogramstatistics

Use this element to specify a column-specific value for the databasedegree (ignored on SQL Server) and the samplingpercentage attributes. By default, PolicyCenter issues a single dbms_stats.gather_table_stats(... 'FOR COLUMNS ...') statement for all columns of interest in the table, including:

- All columns that are the first key column of an index. (Oracle only)
- The retired column, if present.
- The subtype column, if present.
- All columns that have the createhistogram attribute set to true. This is set internally by Guidewire.

If you specify non-default values for either the databasedegree or the samplingpercentage on a particular column, PolicyCenter issues a separate statement for those values alone.

The <histogramstatistics> element has the following format:

```
<histogramstatistics
  name="string"
  numbuckets="integer"
  databasedegree="integer"
  samplingpercentage="integer"/>
```

name specifies a column name. numbuckets controls the maximum number of buckets for the specified histogram. The default value for the number of buckets is 254 for the retired and subtype columns. For all other columns, PolicyCenter uses 75, the database default.

Notes

- For performance reasons, PolicyCenter does not currently create a histogram on publicid columns. These columns are rarely, if ever, referenced in a WHERE clause.
- Also for performance reasons, PolicyCenter tries to combine as many columns as possible into a single statement. Certain tabs in the **Database Catalog Statistics** page display a dbms_stats.gather_table_stats(...'FOR COLUMNS ...') statement with only the associated column for each histogram, regardless of the parameter values. This enables you to specify the most granular statement if a given histogram is out of date.

Configuring Number of Threads for Statistics Generation

PolicyCenter uses the work queue mechanism for statistics generation. You can therefore configure work queue and worker attributes for statistics generation in the work-queue.xml file. Access this file in Guidewire Studio at **configuration** → **config** → **workqueue**. Set parameters for the work queue with workQueueClass set to com.guidewire.pl.system.database.dbstatistics.DBStatisticsWorkItemWorkQueue. For example:

```
<work-queue
   workQueueClass="com.guidewire.pl.system.database.dbstatistics.DBStatisticsWorkItemWorkQueue"
   progressinterval="86400000">
   <worker instances="5" batchsize="10"/>
</work-queue>
```

In this example, the statistics generation work queue is configured to use five worker threads and for each worker to check out ten work items at a time.

After you edit work-queue.xml, you must rebuild and redeploy PolicyCenter.

For more information about work queue configuration, see “Configuring Work Queues” on page 105.

Checking the Database Statistics Updating Process

To check on the state of the process that updates database statistics, use the following command:

```
system_tools -password password -getupdatestatsstate
```

Canceling the Database Statistics Updating Process

To cancel the process that updates database statistics, use the following command:

```
system_tools -password password -cancelupdatestats
```

The database statistics updating process can be paused just as with other work queues. Use the **Work Queue Info** page to pause an in-progress work queue.

Purging Old Workflows and Workflow Logs

Each time PolicyCenter creates an activity, the activity is added to the pc_Workflow, pc_WorkflowLog and pc_WorkflowWorkItem tables. Once a user completes the activity, PolicyCenter sets the workflow status to completed. The pc_Workflow, pc_WorkflowLog and pc_WorkflowWorkItem table entry for the activity are never used again. These tables grow in size over time and can adversely affect performance as well as waste disk space. Excessive records in these tables also negatively impacts the performance of the database upgrade.

Remove workflows, workflow log entries, and workflow items for completed activities to improve database upgrade and operational performance and to recover disk space.

PolicyCenter includes work queues to purge completed workflows and their logs that are older than a configurable number of days. Guidewire recommends that you purge completed workflows and their logs periodically. This reduces performance issues caused by having a large number of unused workflow log records.

To set the number of days after which the `purgeworkflows` process purges completed workflows and their logs, set the following parameter in `config.xml`:

```
<param name="WorkflowPurgeDaysOld" value="value" />
```

Set the value to an integer. By default, `WorkflowPurgeDaysOld` is set to `60`. This is the number of days since the last update to the workflow, which is the completed date.

You can launch the Purge Workflows batch process from the `PolicyCenter/admin/bin` directory with the following command:

```
maintenance_tools -password password -startprocess PurgeWorkflows
```

You can also purge only the logs associated with completed workflows older than a certain number of days. Run the `purgeworkflowlogs` process instead. This process leaves the workflow records and removes only the workflow log records. The `purgeworkflowlogs` process is configured using the `WorkflowLogPurgeDaysOld` parameter rather than `WorkflowPurgeDaysOld`.

You can launch the Purge Workflow Logs batch process from the `PolicyCenter/admin/bin` directory with the following command:

```
maintenance_tools -password password -startprocess PurgeWorkflowLogs
```

Resizing Columns

After the PolicyCenter database is in use, you might discover that you need to change the size of certain columns, such as making a column name longer. PolicyCenter does not provide an automated way of doing this. However, you can follow a commonly used procedure for database changes such as this.

To resize columns

1. Shut down PolicyCenter.
2. Alter the table and add a new temporary column that is the new size.
3. Copy all of the data from the source column to the temporary column.
4. Alter the table and drop the source column.

   Depending on the database, you might need to set the data in this column to all nulls before you can drop the column.
5. Alter the table and add the new source column that is the new size.
6. Copy the data from the temporary column to the new source column.
7. Alter the table and drop the temporary column.
8. Restart PolicyCenter.

IMPORTANT Guidewire does not support resizing base columns.

chapter 4

Data Change API

This topic describes a tightly constrained system for updating data on a running production server other than through PCF pages or web services.

> **WARNING** Use the data change API under extraordinary conditions, with great caution, and upon advice of Guidewire Customer Support. Before registering a data change on a production server, register and run the data change on a development server. Guidewire recommends multiple people review and test the code and the results before attempting on a production server.

This topic includes:

Data Change API Overview

In typical conditions, PolicyCenter data changes in the database using the following techniques:

- Users change data through the user interface, defined by PCF pages
- External systems change data through specific integrations exposed as web services

There may be a need to change production data in a way that had not been predicted enough to define PCF pages or web services for the situation. In typical cases, you can write a new web service or other integration to satisfy your integration need. However, in rare cases there may not be an opportunity to bring your production server down for this improvement to the application.

PolicyCenter provides a tightly constrained system for updating data on a running production server. Because it allows arbitrary execution of data, the ability to create and run code on a production server must be carefully controlled.

WARNING Use the data change API under extraordinary conditions, with great caution, and upon advice of Guidewire Customer Support. Before registering a data change on a production server, register and run the data change on a development server. Guidewire recommends multiple people review and test the code and the results before attempting on a production server.

Separation of Roles

To decrease security risks, the data change API separates its action into two separate tasks with different permissions and entry points:

- **Registering code** – To register the data change code, use either a command line tool (`data_change.bat`) or a WS-I web service (`DataChangeAPI`). The authenticated user must have the permission `wsdatachangeedit`.
- **Running code** – Administrators of PolicyCenter use special administration pages in the application user interface to run the data change code. To view the data change page, the user must have the `admindatachangeview` permission. To actually run the script, the user must have the `admindatachangeexec` permission.

By having two different paths and two different roles, there is no single point of attack.

IMPORTANT Guidewire recommends that you force separation of responsibilities into two different PolicyCenter users. Give each user `wsdatachangeedit` (to register the code) or `admindatachangeexecd` permission (to run it), but not both.

Preserving Results

ClaimCenter captures the results of script execution. This increases accountability and makes debugging easier.

Replay Prevention

To prevent replay attacks, the Data Change API runs each registered script a maximum of one time. If you need to run it again, you must first re-register the script and create a new change control reference.

Typical Use of the Data Change API

There are several steps in using the data change API:

- “Write Data Change Code” on page 50
- “Register a Data Change” on page 51
- “Run Data Change Code” on page 52

Write Data Change Code

You must write Gosu code that correctly and safely makes only the necessary data changes and persists the changes to the database.

WARNING Carefully test your data change code. Guidewire strongly recommends that multiple people review and approve the code for safety and correctness before proceeding.

To persist changes to the database, use the `gw.Transaction.Transaction` class and its method `runWithNewBundle`. See "Running Code in an Entirely New Bundle" on page 345 in the *Gosu Reference Guide*. You pass the method a block that runs code. If the block does not throw an exception, PolicyCenter persists any data changes from your Gosu block to the database. If the block throws an exception, no changes persist to the database.

Use data change Gosu APIs to configure logging within data change code. These APIs generate logging information that users can see in human-readable output in data change user interface:

- To log entity field-level entity changes, call `DataChange.util.setDetailResultWriting(bundle)`. The logging information includes information about added objects, deleted objects, and field-level changes on every object. For updated properties, the logging information includes each field value before the change and after the change.
- To log arbitrary text data, call `DataChange.util.ResultsWriter`. That property returns an appender, which is an object that implements the interface `java.lang.Appendable`. That object has several method signatures of the `append` method. The simplest method signature takes a `CharSequence` object, such as a standard `String` object.

For example, the following code uses the `setDetailResultWriting` method and the `ResultsWriter` property:

```
gw.transaction.Transaction.runWithNewBundle(\ bundle -> {

  // For demonstration, get a User object and make minor data change to the first name
  var u = gw.api.database.Query.make(User).select().first()
  bundle.add(u)
  u.Contact.FirstName = u.Contact.FirstName + "SUFFIX"

  // Determine what you want to write to the data change log
  var msg = "For PublicID '${u.PublicID}' User.DisplayName is now '${u.DisplayName}'!"

  // To log arbitrary text in Data Change UI, get a results writer (type is java.lang.Appender)
  var rw = DataChange.util.ResultsWriter
  rw.append("Add arbitrary log message here\n")

  // enable detailed logging of each property value before and after our change
  DataChange.util.setDetailResultWriting(bundle)

  // for testing in Studio Scratchpad, also print to standard console
  // print("To console: " + msg)

})
```

To test and debug your code in Studio Scratchpad, you may want to print to the console using the standard `print` statement. Also, add one more argument to the `runWithNewBundle` method to represent a user name. For example, pass the `String` value `"su"` to make your writable bundle as the super user.

Design your data change code to minimize the number of entity instances you change. Too many changes in entity data increases the chance of memory issues or concurrent data exceptions.

Save your Gosu code to a local file that ends in `.gsp`.

Register a Data Change

There are two ways to register your data change code

- The command line tool `PolicyCenter/admin/data_change.bat`
- The WS-I web service `DataChangeAPI`

The data change registration details vary between these two variants.

In all cases, before proceeding you must have:

- Data change code in the form of a Gosu script that you have already tested in your development environment. See "Write Data Change Code" on page 50.
- A human-readable description for your data change
- A unique reference ID that you create to represent this data change

Register a Data Change From Command Prompt

WARNING Use the data change API under extraordinary conditions, with great caution, and upon advice of Guidewire Customer Support. Before registering a data change on a production server, register and run the data change on a development server. Guidewire recommends multiple people review and test the code and the results before attempting on a production server.

To register a data change from command prompt

1. Open a command prompt.
2. Set your working directory to `PolicyCenter/admin`.
3. Run the following command:

```
data_change.bat -description DESCRIP -edit REFID -gosu PATH -server SERVERURL -user USER -password PW
```

For example:

```
data_change.bat -description "Fix Employee Name"
   -edit REFID_1234 -gosu c:\PolicyCenter\datachange\gosudatachange_REFID1234.gsp
   -server http://TESTINGSERVER:8080/pc -user su -password gw
```

For complete documentation on all command line options, see “Data Change Command Line Reference(data_change.bat)” on page 53.

The script outputs results such as:

```
Running data_change.gsp
Connecting as su to URL http://localhost:8080/cc/ws/gw/webservice/systemTools/DataChangeAPI
Edit change ref=REFID1234 publicId=cc:1
```

Register a Data Change From a Web Service

WARNING Before registering a data change on a production server, register and run the data change on a development server. Guidewire recommends multiple people review and test the code and the results before attempting on a production server.

If you do not want to use a command line tool, you can register a data change with the WS-I web service `DataChangeAPI`. The command line tool `data_change` works by calling the `DataChangeAPI` web service on a running PolicyCenter server. For more information about the related command line tool, see:

- “Register a Data Change From Command Prompt” on page 52
- “Data Change Command Line Reference(data_change.bat)” on page 53

To register a data change, call the `DataChangeAPI` web service method `updateDataChangeGosu`. Pass the method the reference ID, a human-readable description, and the Gosu code to run. Pass all of these arguments as `String` objects. The method returns the public ID of the new `DataChange` entity instance.

For example:

```
var gosuScript = "gw.transaction.Transaction.runWithNewBundle(\ bundle -> {
                    print(""DATA CHANGE!"") })"

var publicID = datachangeAPI.updateDataChangeGosu("REFID_1234",
               "Fix for Issue 1234 regarding missing Employee ID", gosuScript)
```

Run Data Change Code

WARNING Use the data change API under extraordinary conditions, with great caution, and upon advice of Guidewire Customer Support. Before registering a data change on a production server, register and run the data change on a development server. Guidewire recommends multiple people review and test the code and the results before attempting on a production server.

To run a data change

1. Confirm that someone created and registered a data change as described in "Write Data Change Code" on page 50. You must know the reference ID for the registered data change.
2. Log into PolicyCenter as an administrator. Note that the user that created and registered the data change may not be the same person as the person running the data change in the production environment. You can define these roles to have different permissions.

   To view the data change page, the user must have the `admindatachangeview` permission. To actually run the script, the user must have the `admindatachangeexec` permission.

   IMPORTANT Guidewire recommends that you force separation of responsibilities into two different PolicyCenter users. Give each user `wsdatachangeedit` (to register the code) or `admindatachangeexecd` permission (to run it), but not both.

3. Navigate to **Admin** → **Utilities** → **Data Change**.
4. In the list of data changes, use the **Reference** column to find the data change request by its reference ID. Click on the data change row in that list. If the list is long, you can use the picker on the page to filter to just ones with status **Open**.
5. The page displays the Gosu code for that data change. Review the Gosu code to confirm it is what you expect.
6. Click **Execute**.
7. The page may not display the results immediately in the **Result** pane. The status may appear as the status **Executing** in the list of data changes. After some amount of time, click reload on the page to view the current status and results.

   If the change is successful, it confirms in a message that uses your reference ID:

   ```
   REFID1235 finished okay
   ```

   If there are compile errors or exceptions, they appear in the user interface in the **Result** pane.
8. Confirm your changes in the database and check your logging results from the change.

   WARNING Consult with other people as needed to determine that the data change is safe and correct.

   If you need to re-run a successful data change, you must first re-register the script with a new reference ID. This is a requirement preserves the integrity of the results log. See "Register a Data Change" on page 51.
9. If the data change appears safe in your development environment, carefully register and run the data change on the production server.

   WARNING Use the data change API under extraordinary conditions, with great caution, and upon advice of Guidewire Customer Support. Before registering a data change on a production server, register and run the data change on a development server. Guidewire recommends multiple people review and test the code and the results before attempting on a production server.

Data Change Command Line Reference(data_change.bat)

The `data_change.bat` command has the following options. Use exactly one data change action argument. Always add all server authentication options.

See also

- "Data Change API Overview" on page 49
- "Typical Use of the Data Change API" on page 50

- “Data Change Web Service Reference (DataChangeAPI)” on page 54

Option for data_change.bat	Description
Data change actions (choose one)	
`-edit` *REFID* `--edit` *REFID*	This option indicates you want to create a new data change or edit an existing data change. Include a unique reference ID for this data change. If the data change succeeded with no compile errors, you cannot edit it. You must re-register the script with a new reference ID. If the data change was never run, or had compile errors, you can update (edit) the Gosu code with the same reference ID. If you use the `edit` option: • include the `gosu` argument to include your Gosu data change code • include the `description` argument for a description
`-discard` *REFID* `--discard` *REFID*	This option indicates you want to discard a data change that you already registered. Pass a data change reference ID. You cannot discard a data change that was already run.
`-result` *REFID* `--result` *REFID*	This option indicates you want the result of a data change that you already registered. Pass a data change reference ID. If a user attempted to run it and there were parse errors, the results include the errors.
`-status` *REFID* `--status` *REFID*	This option indicates you want the status of a data change that you already registered. Pass a data change reference ID. The tool prints the status, which is `Open`, `Discarded`, `Executing`, `Failed`, or `Completed`.
Data	
`-description` *DESCRIPTION* `--description` *DESCRIPTION*	A human-readable description of the change. Include this option when you use the `edit` argument. For testing, the description is optional. For production use, include the description. Put quotes around the description to permit space characters in the description.
`-gosu` *FILEPATH* `--gosu_file` *FILEPATH*	The full path name to a Gosu script. Include this option when you use the `edit` argument. You can use a full path name, or a relative path that is relative to the current working directory.
Server authentication (required)	
`-server` *SERVER_URL* `--server` *SERVER_URL*	The server URL, including port number and web application name. For example: `http://servername:8080/pc`
`-user` *USER* `--user` *USER*	The user to use for this request. The user must have the permission `wsdatachangeedit`
`-password` *PASSWORD* `--password` *PASSWORD*	The user password

Data Change Web Service Reference (DataChangeAPI)

To register a data change, or to check status on it, use methods on the `DataChangeAPI` web service.

See also

- “Data Change API Overview” on page 49
- “Typical Use of the Data Change API” on page 50

- “Data Change Command Line Reference(data_change.bat)” on page 53

DataChangeAPI method	Description
`updateDataChangeGosu`	Register a data change. Pass the method the reference ID, a human-readable description, and the Gosu code to run. Pass all of these arguments as `String` objects. The method returns the public ID of the new `DataChange` entity instance. If the data change succeeded with no compile errors, you cannot edit it. You must re-register the script with a new reference ID. If the data change was never run, or had compile errors, you can update (edit) the Gosu code with the same reference ID.
`discardDataChange`	This option indicates you want to discard a data change that you already registered. Pass a data change reference ID as a `String`. You cannot discard a data change that was already run.
`getDataChangeResult`	This option indicates you want the result of a data change that you already registered. Pass a data change reference ID. It returns a `String` that represents the results in the `DataChange` entity instance. If a user attempted to run it and there were parse errors, the results include the errors.
`getDataChangeStatus`	This option indicates you want the status of a data change that you already registered. Pass a data change reference ID. The method returns a `DataChangeStatus` typecode. Values include `Open`, `Discarded`, `Executing`, `Failed`, `Completed`.

chapter 5

Managing PolicyCenter Servers

This topic discusses the PolicyCenter application server, run levels, modes, monitoring servers, and application server caching.

This topic includes:

Stopping the PolicyCenter Application

Before you stop the PolicyCenter application, stop all work queues. Distributed workers run on daemon threads. When the JVM (Java Virtual Machine) exits, these threads are destroyed. This can cause issues if a thread is destroyed while processing a work item. Also, work queues can make calls to plugins. You could implement a plugin that makes a blocking call to an external system or otherwise take a long time to return. If you do not shut down worker threads correctly, you could end up with inconsistent data.

You can stop work queues from the PolicyCenter **Batch Process Info** page.

To stop work queues from the Batch Process Info page

1. Press `ALT+SHIFT+T` to display the **Server Tools** tab.

2. Click **Batch Process Info** if not already on the **Batch Process Info** page.
3. For any process that has a **Status** of **Running**, click the **Stop** button in the **Action** column. Wait for all processes to have a **Status** of **Inactive** before stopping PolicyCenter.
4. For any process that has a **Next Scheduled Run** time that is before the time that you will stop PolicyCenter, click **Stop** in the **Schedule** column. This disables the schedule for the current PolicyCenter session. The schedule is enabled again when you restart PolicyCenter, according to the settings in `scheduler-config.xml`. See "Scheduling Work Queues and Batch Processes" on page 107.

After you have stopped all work queues and disabled the schedule for upcoming work queues, you can stop the PolicyCenter application. To stop PolicyCenter in a production environment, stop the application server. To stop PolicyCenter in a development environment, run the `gwcc dev-stop` command from `PolicyCenter/bin`.

Server Modes and Run Levels

The PolicyCenter server can run in development, test or production mode. The mode determines available functionality at various run levels of the server.

With all application server types except QuickStart, PolicyCenter can run in either development, test or production mode. Only development mode is available with QuickStart.

Test mode is identical to production mode with the following exceptions while running in test mode:

- You can adjust the testing system clock by using the `setCurrentTime` method on the `ITestingClock` plugin. See "Testing System Clock" on page 155. Also see "Testing Clock Plugin (Only For Non-Production Servers)" on page 254 in the *Integration Guide*.
- Both the **Server Tools** and **Internal Tools** tabs are available. In production mode, only the **Server Tools** tab is available. For information about these tools, see "Using Server and Internal Tools" on page 133.
- PolicyCenter prints a message to the console during startup indicating that the server is running in test mode.
- The browser title bar for a browser connected to PolicyCenter indicates that PolicyCenter is in test mode.

Other than the exceptions listed, test mode is identical to production mode, so this document does not describe test mode separately from production mode.

Guidewire provides these modes as a safety precaution so that development tools are not used on a production server. Some system functions are useful for development, but are not appropriate, or even dangerous, if used in a production environment. In development and test mode, both the **Server Tools** and **Internal Tools** tabs are available. In production mode, only the **Server Tools** tab is available. See "Using Server and Internal Tools" on page 133.

An example is the `ITestingClock` plugin that provides functionality for setting the current time and is critical for testing time-sensitive processes. You can also use this plugin to modify the current time in a running server for demonstrations. However, use of this plugin in a production environment could have disastrous results. Therefore, you can only use this plugin when the server is in development or test mode. Test mode is identical to production mode except that in test mode you can adjust the testing system clock. See "Testing Clock Plugin (Only For Non-Production Servers)" on page 254 in the *Integration Guide*. See also "Testing System Clock" on page 155.

Another difference between modes in PolicyCenter is that certain types of product model changes cannot be deployed to a production environment. For more information on what changes are illegal, see "Preventing Illegal Product Model Changes" on page 111 in the *Product Model Guide*. These locks are in place to protect data integrity. Any time the database must be preserved (and upgraded), put the server into production or test mode to minimize the risk of data corruption.

By default, PolicyCenter starts in production mode on all supported application servers other than the QuickStart server. PolicyCenter on the QuickStart server always runs in development mode. You cannot run PolicyCenter on the QuickStart server in production or test mode.

The PolicyCenter server can also be put into `MULTIUSER`, `DAEMONS`, and `MAINTENANCE` run levels. See "Using the Maintenance Run Level" on page 61. These run levels are independent of the mode. The combination of mode and run level determines the availability of functionality, such as the user interface and web services. For details, see "Server Modes and Run Levels" on page 58.

The following table shows which functionality is available for the possible combinations of modes and run levels.

System Run Level	Simplified Run Level	Production mode	Development mode
`MULTIUSER`	multiuser	User interface available. All logins allowed. **Server Tools** available for users with admin permission only. **Internal Tools** not available. Web services available. Product Model checked for illegal changes.	User interface available. All logins allowed. Studio connection allowed. Server Tools available to all users if `EnableInternalDebugTools` is set to `true` in `config.xml` **Internal Tools** available. Web services available.
`DAEMONS`	daemons	User interface not available. Web services available. Product Model checked for illegal changes. Work queues (including workflow) available. Workflow Stat Manager available. Scheduler available. Daemons started by application, such as QPlexor (for messaging) available.	User interface available. All logins allowed. Web services available. Work queues (including workflow) available. Workflow Stat Manager available. Scheduler available. Daemons started by application, such as QPlexor (for messaging) available.
`NODAEMONS`	maintenance	User interface not available. Web services available. Staging table loading available. Product Model checked for illegal changes. Batch processes available.	User interface available. All logins allowed. Web services available. Staging table loading available. Batch processes available.
`SHUTDOWN`	Reported as starting	User interface not available. Web services not available. Database not available.	User interface not available. Web services not available. Database not available.
`GUIDEWIRE_STARTUP`	Reported as starting	User interface not available. Web services not available. Database not available.	User interface not available. Web services not available. Database not available.
`NONE`	Reported as starting	Nothing available.	Nothing available.

Setting the Server Mode

You can change the server mode while using any application server type except QuickStart. The QuickStart server always runs PolicyCenter in development mode.

Control the mode through the system parameter:

```
-Dgw.server.mode=(dev|prod|test)
```

To change the mode, restart the server and set the `-Dgw.server.mode` parameter to `dev`, `test` or `prod`:

```
-Dgw.server.mode=dev
```

or

```
-Dgw.server.mode=test
```

or

```
-Dgw.server.mode=prod
```

PolicyCenter ignores this parameter on the QuickStart server.

Determining Server Mode

You can determine the PolicyCenter mode by reading the console log as you start PolicyCenter or by checking the browser title bar.

Note: Whenever the server starts in development mode, PolicyCenter logs a warning.

Setting the Server Run Level

You can set the server run level to multiuser, daemons, or maintenance by using the `system_tools` command in `PolicyCenter/admin/bin`. The following examples show how to set the run level.

To set the server run level to multiuser

```
PolicyCenter/admin/bin/system_tools -password password -multiuser
```

To set the server run level to daemons

```
PolicyCenter/admin/bin/system_tools -password password -daemons
```

To set the server run level to maintenance

```
PolicyCenter/admin/bin/system_tools -password password -maintenance
```

You can also set the server run level using the `SystemToolsAPI` web service. You cannot set the server to the `SHUTDOWN`, `GUIDEWIRE_STARTUP` or `NONE` run level. However, `SystemToolsAPI.getRunlevel()` can report these run levels. See “Getting and Setting the Run Level” on page 97 in the *Integration Guide*.

If you run PolicyCenter in a clustered environment, you cannot place all the computers in a particular mode with a single command. Instead, you must run the command individually on each computer.

Determining the Server Run Level

You can determine the server run level by using the `system_tools` command in `PolicyCenter/admin/bin`. The following example shows how to determine the run level.

```
PolicyCenter/admin/bin/system_tools -password password -ping
```

The returned message indicates the server run level. The possible responses are:

- `MULTIUSER`
- `DAEMONS`
- `MAINTENANCE`
- `STARTING`

You can also determine the server run level by directly calling the API `SystemToolsAPI.getRunlevel()`.

You can also determine the server run level from an unauthenticated web page. See “Checking Server Run Level” on page 83.

Using the Maintenance Run Level

Periodically, you need to perform maintenance on PolicyCenter, such as importing new security roles. To prevent users from logging into PolicyCenter during these activities, place PolicyCenter into the maintenance run level. Use the following command:

```
PolicyCenter/admin/bin/system_tools –password password –maintenance
```

The maintenance run level effectively disables the PolicyCenter web interface if the server is in production mode. PolicyCenter stops allowing new user connections and halts existing user sessions to production mode instances while running at the maintenance run level.

PolicyCenter still allows connections made through APIs or command line tools for any daemons with a minimum run level equal or lower than `NODAEMONS`. This permits integration processes to proceed without interference from unplanned activities by non-administrator users.

Server Startup Tests

The PolicyCenter server performs a series of tests during startup. Some of these tests prevent the server from starting. Other tests are warnings and allow the server to start. These checks warn about potential problems with the graphs that might not be an issue depending on business logic.

See “Domain Graph Validation” on page 255 in the *Configuration Guide*.

Monitoring the Servers

You can use an HTTP ping utility provided by Guidewire to check server status. See “Checking Node Health” on page 81.

Use standard operating system tools to monitor memory usage, CPU usage, and disk space to verify that the servers run smoothly. In particular, monitor disk space for log files, so PolicyCenter does not run out of disk space for logs. Archive and truncate system logs periodically to prevent the PolicyCenter logs from growing too large.

If the server crashes with the following JVM error, increase the maximum heap size (`-Xmx` setting) of the JVM.

```
Internal Error (53484152454432554E54494D450E43505001A8)
```

See “Configuring the Application Server” on page 17 in the *Installation Guide* and documentation provided with the application server for instructions on increasing the maximum heap size.

Monitoring and Managing Event Messages

PolicyCenter generates a large number of events. In a typical company’s environment, it can be necessary or helpful for PolicyCenter to notify other applications of these events through an integration. PolicyCenter integration developers create message destination objects that provide the means for passing information between PolicyCenter and a particular destination. Rule writers can write Gosu rules to generate messages in response to events of interest. PolicyCenter queues these messages and dispatches them to receiving systems by using the destination objects.

Monitor message traffic to ensure that the integration is running smoothly. This section discusses topics in monitoring and managing event messages. For more information about messages, including how to create message destination objects, see “Messaging and Events” on page 289 in the *Integration Guide*.

How PolicyCenter Processes Messages

Every time PolicyCenter sends an event message, it expects to receive a positive acknowledgement (ack) back from the destination indicating it received and processed the message. PolicyCenter retains completed messages until you purge them. Since the number of messages in PolicyCenter can grow to be large, purge completed messages on a regular basis. Use the following command:

```
messaging_tools -password password -purge MM/DD/YY
```

For example:

```
messaging_tools -password gw -purge 02/06/06
```

In this example, this command purges all completed messages received prior to `02/06/06`.

Messages can have several different statuses. The following table describes the different message statuses and what they mean:

Unsent	The message has not been sent. The message might be waiting on a prior message. Or, the destination might not be processing messages because it is suspended. Or, the destination is falling behind. PolicyCenter can generate messages very quickly
Needing Retry	Waiting to attempt a retry. PolicyCenter attempted to send the message but the destination threw an exception. If the exception was retryable, PolicyCenter automatically attempts to retry the send before turning the message into a failure. PolicyCenter attempts to send an event message several times. Typically, you can configure the number of retries and the interval between them for an integration. Review documentation for the specific destination to find out how to configure it.
In Flight	PolicyCenter is waiting for an acknowledgement.
Messages Failed	A message can fail for several reasons. • A processing error, the destination did not process the message successfully. • The destination returns a negative acknowledge (nack) indicating that the message failed. • The message was embedded in a series of messages, one or more of which failed.

If PolicyCenter receives an unrecoverable or unexpected exception from a send attempt, or reaches the retry limit, it does not send messages to that destination until you clear the error. If PolicyCenter receives a processing error that is not retryable, PolicyCenter also suspends the destination and waits for you to clear the error. To clear an error, either manually retry the send or skip the message. Do this from the user interface or from the command line.

If PolicyCenter becomes completely out of synchronization with an external system, such that skipping or retrying a message is insufficient to resynchronize the two systems, resynchronize the entire destination. A resynchronization causes PolicyCenter to drop all pending and failed messages and resend all the messages associated with a particular claim.

Working with the Destinations Page

PolicyCenter lists each message destination in the **Event Messages** page. You access this page from the **Administration** tab by choosing **Event Messages**. The first page of **Event Messages** contains cumulative information about message destinations.

You can select a message destination and **Suspend** or **Resume** the individual synchronization. You can also restart the messaging engine within PolicyCenter itself.

If a destination is running correctly, you do not see any accumulation of information in the columns. If there is a problem and messages begin to accumulate, you can drill down into a message destination by clicking the destination name. This opens the **Destination** page. From this page, you can see additional detail about any clog in a destination. This information can assist you in diagnosing the error, in particular you can use the **Error Message** column to see the possible cause of a particular clog.

The **Destination** page lists all failed or in-process messages for an account for all destinations. You can search for a particular account to respond to a query from an end user and then open the account's detail view. From this page, you can select one or more accounts and indicate that the failed or in-flight message for each account be skipped, retried, or resynced.

Configuring Message Destinations

You create and configure message environments and destinations in the `messaging-config.xml` file. Access `messaging-config.xml` in Guidewire Studio at **configuration** → **config** → **Messaging**. See "Messaging Editor" on page 131 in the *Configuration Guide*.

Tuning Message Handling

A PolicyCenter server reads integration messages from a queue and dispatches them to their destinations. However, there is no guarantee that messages in the queue are ready for dispatching in the same order in which PolicyCenter places the messages in the queue.

For example, the server can start writing `message1` to the queue, and then start writing `message2` to the same queue. It is possible that the server completes and commits `message2` while still writing `message1`. This does not, in itself, present an issue. However, if the server attempts to read messages off the queue at this moment, then it skips the uncommitted `message1` and reads `message2`. You are most likely to encounter this situation in a clustered PolicyCenter environment.

To address this situation, PolicyCenter provides the `IncrementalReaderSafetyMarginMillis` parameter in the `config.xml` file. This determines how long after detecting a skipped message that PolicyCenter attempts to read messages again. This waiting period gives the skipped message a chance to be committed. If the message has not been committed after waiting this long, then PolicyCenter assumes the message is lost and will never be committed, and PolicyCenter skips the message permanently.

For example, in the previous scenario, PolicyCenter waits 10 seconds (the default parameter value) before attempting to read messages again, beginning with the skipped `message1`. If `message1` has still not been committed at that time, PolicyCenter skips it permanently.

Set the `IncrementalReaderSafetyMarginMillis` parameter long enough to give messages time to be committed without prematurely marking them as permanently skipped. However, because no other messages are read during this waiting period, do not set `IncrementalReaderSafetyMarginMillis` so long as to delay the delivery of messages. You can also set the `IncrementalReaderPollIntervalMillis` and `IncrementalReaderChunkSize` parameters to configure the message reading environment.

System Users

PolicyCenter creates system users in addition to the standard users who log in to PolicyCenter.

`"Temporary system user"` is the name given to an unauthenticated user session. PolicyCenter creates such sessions for login. By definition, there is no user associated with the login screen. The `system_tools -sessioninfo` command does not filter out this user. The **Management Beans** page does filter out this user.

PolicyCenter also uses `sys`, the system user. This is the user PolicyCenter uses to do automated work such as running batch processes. Each time PolicyCenter needs to do such work, it creates a session with the `sys` user. This is why there might appear to be many sessions with the `sys` user. Session in this sense is not a web session. Rather, it represents the authentication of a user. While PolicyCenter authenticates a user, PolicyCenter creates one of these sessions to represent that.

Configuring Minimum and Maximum Password Length

The `MinPasswordLength` and `MaxPasswordLength` parameters in `config.xml` control the minimum and maximum number of characters for passwords. For example, if you want all users in your system to have a password length of at least six characters and a maximum of sixteen, set the following in `config.xml`:

```
<param name="MinPasswordLength" value="6"/>
<param name="MaxPasswordLength" value="16"/>
```

Configuring Client Session Timeout

PolicyCenter creates a session for each browser connection. PolicyCenter uses the application server's session management capability to manage the session. Each session receives a security token that PolicyCenter server preserves across multiple requests. The server validates each token against an internal store of valid tokens.

You can configure the timeout value for a session by setting the value of the following property in `config.xml`:

```
<param name="SessionTimeoutSecs" value="10800"/>
```

Typically, the application server determines the session timeout value according to the following hierarchy.

Level	Description
Server	The session timeout to use for all applications on the server if a timeout value is not specified at a higher level.
Enterprise application	The session timeout specified at the enterprise application level. You can specify this value at the EAR file level. You can set the enterprise application session timeout value to override the server session timeout value.
Web application	The session timeout specified at the web application level. You can specify this value at the WAR file level.You can set the web application session timeout value to override the enterprise application and server session timeout values.
Application level	The session timeout specified in the application `web.xml` file. PolicyCenter does not specify a session timeout in `web.xml`.
Application code	An application can override any other session timeout value. PolicyCenter uses the session timeout value specified by the `sessiontimeoutsecs` parameter in `config.xml`.

Avoiding Session Replication

PolicyCenter does not implement user session replication for various reasons. Do not attempt to replicate sessions across nodes or persist user sessions, for the following reasons:

- PolicyCenter sessions are not serializable. Therefore, you cannot replicate a PolicyCenter session, either with or without persistence to the database.
- PolicyCenter sessions hold the user state in memory and contain a lot of information. Guidewire estimates that this amounts to 1 MB of data on average for a 32-bit application server and close to 2 MB for a 64-bit server. Replication would create significant cross-node communication that is detrimental to performance.
- PolicyCenter commits changes to the database on almost all transactions. Noticeable exceptions are some wizards for which PolicyCenter commits data changes only after the user completes all necessary entries.
- PolicyCenter scales horizontally almost linearly. The implementation of a session replication solution would very likely impede that linear scalability.

An application server node failure can result in loss of changes recently entered into the browser, loss of in-flight write transactions or, at worst, loss of wizard entries. Integrate PolicyCenter with a single sign-on solution to prevent users from having to log back in at failover.

Application Server Caching

Guidewire implements an object caching mechanism at the application server layer. This mechanism limits reads to the database, thereby significantly improving performance.

This topic includes:

Cache Management

Objects do not remain forever present or valid in the cache. Several mechanisms exist to ensure that cache entries remain relevant:

- A stale timeout mechanism ensures that the server does not use excessively old object entries. An object is stale if it has not been refreshed from the database within a configurable amount of time. Upon accessing a cache entry, the server calculates the duration since the object was last read from the database. If that duration exceeds the stale time, the server refreshes the cache entry from the database. To avoid increased complexity, PolicyCenter prefers this mechanism over evicting objects upon stale timeout. You can set a default stale time by adjusting the `GlobalCacheStaleTimeMinutes` parameter in `config.xml`.
- An evict timeout mechanism removes old objects from the cache. For example, a bean has an evict time of 15 minutes and a stale time of 30 minutes. If the server uses the object once every 14 minutes, PolicyCenter never evicts the cache entry, but the entry does eventually become stale. You can set the default evict time by adjusting the `GlobalCacheReapingTimeMinutes` parameter in `config.xml`. `GlobalCacheReapingTimeMinutes` is initially set to `15` minutes. The minimum value for this parameter is `1` minute. The maximum value for this parameter is the smaller of `120` and `GlobalCacheStaleTimeMinutes`.
- Upon reception of an inter-cluster message indicating that an object value was changed in another node, the server marks the corresponding entry in the cache obsolete and available for reuse.

The importance of these mechanisms becomes more meaningful as other aspects of the cache are described.

Caching and Stickiness

The cache mechanism is fully leveraged if users return to the same node across different HTTP requests. In a clustered environment, the load balancer must direct requests to the same application server node upon consecutive interactions. This mechanism, referred to as "stickiness", enables a true horizontal scalability solution. For more information on load balancing options for PolicyCenter, consult Guidewire Services.

Each application server manages its own cache. It is possible for the same object to live in the cache of two or more application servers at the same time. Some object sets, such as users, likely live in the global cache of all application servers in a cluster.

Concurrent Data Change Prevention

Different users, either on the same application server instance or on different ones, might change objects concurrently. Guidewire implements a data versioning mechanism to prevent corruption in such cases. As PolicyCenter updates an object value to the database, PolicyCenter compares a counter associated with the object to the counter in the database. A counter value mismatch indicates that the object was concurrently changed. In such

case, PolicyCenter rejects the change and the cache mechanism throws a concurrent change exception. PolicyCenter presents the user who initiated the concurrent change with the error and reloads the latest data. The user can then reapply the changes. Furthermore, PolicyCenter commits changes in an atomic bundle, ensuring transactional integrity. Therefore, PolicyCenter enforces protection against concurrent data changes across the whole transaction. This mechanism is a standard design pattern called optimistic locking.

Concurrent change exceptions occur only if two users modify the same object. A proper organization of the user community avoids this. Nevertheless, if two users modify the same object, any automatic resolution carries a significant risk of causing unwanted modifications. The optimistic locking mechanism causes very few concurrent data change exceptions and users can easily resolve those exceptions.

Other design patterns exist for concurrent data changes. The pessimistic locking pattern prevents all other users from modifying an object while one user or batch process is making a change. In many cases pessimistic locking becomes completely dysfunctional. For example, if a user or batch process cannot complete a change, any other user or batch process is blocked. Pessimistic locking systems generally become impractical. Therefore, PolicyCenter uses the optimistic locking mechanism.

Caching and Clustering

Cluster inter-communication ensures that if an object changes in one node cache, the server sends a cache invalidation message to other application servers. This message instructs the other servers to tag the cache entry for the object as obsolete. The next time a server needs the object, the server reloads the object value from the database. This mechanism is different from full cache synchronization, in which the server broadcasts the new value of the object to other nodes.

Network failures or other issues can disrupt communication between nodes. In such cases, object change messages might be lost. The data versioning mechanism prevents data corruption by raising concurrent data change exceptions. After the disruption is corrected, the cache mechanism functions fully again. Concurrent data exceptions no longer occur.

Performance Impact

This section distinguishes two caches:

- Application server cache: the one described in this section
- Database server cache: a database uses this cache to store data retrieved from storage.

Proper caching behavior is critical to performance. “Analyzing and Tuning the Application Server Cache” on page 67 describes how to size a cache correctly.

The application server cache is purely local to the application server. Therefore, one application server node cache might contain information on a specific object while another node might not contain that information. For example, if a PolicyCenter user works on a policy, PolicyCenter loads corresponding objects on the application server node to which the user connects. If another user must approve the action of the first user, the approver user might interact with another application server node. In that case, the other node likely does not have the corresponding information in cache. Therefore, the approver might experience slower performance as server must populate the cache.

Cache content is lost when you stop the application server node. Therefore, when you start the application server, expect lower performance during a ramp-up phase.

Batch processes leverage the cache mechanism. Batch jobs can work on many objects. Therefore, they can use the cache extensively. This can have the adverse effect of prematurely evicting objects from the cache, thereby forcing additional cache loads. For this reason, if you run many intensive batch processes, consider dedicating a specific application server instance to batch activity with no online traffic directed to it.

Cache Thrashing

Cache thrashing is a phenomenon whereby evictions remove cache entries prematurely and force additional database reloads that are detrimental to performance. There are several cases that can lead to cache thrashing:

- A single data set can be too large to reside in the global cache. This forces the server to load the same data from the database and subsequently evict the data, potentially thousands of times, while loading a single web page. This results in serious performance issues.
- Some concurrent actions result in thrashing. For example, a user logs on to an application server that is functioning as the batch server. A batch job, which can load many objects into the cache, can remove objects from the cache. This forces the server to reload the cache as the user again needs those objects.

If the batch server experiences cache thrashing, dedicate the batch server to batch processing only. In this case, do not have the batch server also handle user requests.

See “Detecting Cache Thrashing” on page 69.

Cache Impact on Memory Utilization

The maximum size of the cache is dependent on cache parameters. See “Analyzing and Tuning the Application Server Cache” on page 67. The application server cache grows in size to reach a maximum specified by cache sizing parameters. Java does not provide a good means to estimate the memory usage of objects. Therefore, the maximum size of a cache cannot be reliability estimated. If the cache size exceeds the maximum heap size, the application eventually runs out of memory.

Larger caches increase memory starvation issues. Larger caches expand the memory footprint of the application. Performance decreases as garbage collection becomes more frequent and analyzes more objects.

Set the cache as large as needed, but no larger. Monitor garbage collection to extrapolate memory usage patterns and garbage collection statistics. See “Analyzing Server Memory Management” on page 70.

Analyzing and Tuning the Application Server Cache

The `config.xml` file contains cache parameters for PolicyCenter. Access this file from Guidewire Studio at **configuration** → **config**.

Parameter	Description
`ExchangeRatesRefreshIntervalSecs`	The number of seconds between refreshes of the exchange rates cache. This is a specialized cache only for exchange rates. See “Special Caches for Rarely Changing Objects” on page 70.
`GlobalCacheActiveTimeMinutes`	Time, in minutes, that PolicyCenter considers cached objects active. The cache gives higher priority to preserving these objects. This can be thought of as the period that items are being heavily used, for example, how long a user stays on a page. Set `GlobalCacheActiveTimeMinutes` to a value less than `GlobalCacheReapingTimeMinutes`.

Parameter	Description
GlobalCacheDetailedStats	Boolean that specifies whether to collect detailed statistics for the global cache. Detailed statistics are data that PolicyCenter collects to explain why items are evicted from the cache. Basic statistics, such as miss ratio, are still collected regardless of the value of GlobalCacheDetailedStats. Disabling collection of detailed cache statistics can sometimes improve performance. GlobalCacheDetailedStats is set to false by default. Set the parameter to true to help tune your cache. If the GlobalCacheDetailedStats parameter is set to false, the **Cache Info** page does not include the **Evict Information** and **Type of Cache Misses** graphs. At runtime, use the **Management Beans** page to enable the collection of detailed statistics for the global cache. See also: • “Management Beans” on page 137 • “Cache Info” on page 142
GlobalCacheReapingTimeMinutes	Time, in minutes, since last use of a cached object before PolicyCenter considers the object eligible for reaping. This can be thought of as the period during which PolicyCenter is most likely to reuse an object. An evict timeout mechanism removes old objects from the cache. Once per minute, a thread evicts cache entries that have not been used for a period equal to or greater than GlobalCacheReapingTimeMinutes. This mechanism differs from the stale timeout mechanism. The stale timeout mechanism refreshes from the database those cache entries that have exceeded the stale time. This process occurs as the server accesses a cached object. The evict timeout mechanism deletes any cache entries that are older than the default evict time. An object can become stale but not evicted if it is continually in use. For example, a bean has an evict time of 15 minutes and a stale time of 30 minutes. If the server uses the object once every 14 minutes, PolicyCenter never evicts the cache entry, but the entry does eventually become stale. GlobalCacheReapingTimeMinutes is initially set to 15 minutes. The minimum value for this parameter is 1 minute. Since the eviction thread only runs once per minute, a smaller value would not make sense. The maximum value for this parameter is 15 minutes.
GroupCacheRefreshIntervalSecs	The number of seconds between refreshes of the groups cache. This is a specialized cache only for groups. See “Special Caches for Rarely Changing Objects” on page 70.
GlobalCacheSizeMegabytes	Maximum amount of heap space used to store cached entities, expressed as a number of megabytes. This parameter supersedes the value of GlobalCacheSizePercent. At runtime, you can use the **Cache Info** or **Management Beans** page to modify this value. See also: • “Cache Info” on page 142 • “Management Beans” on page 137
GlobalCacheSizePercent	Maximum amount of heap space used to store cached entities, expressed as a percentage of the maximum heap size.

Parameter	Description
GlobalCacheStaleTimeMinutes	Time, in minutes, after which PolicyCenter considers an object in the cache stale if it has not been refreshed from the database. A stale timeout mechanism ensures that the server does not use excessively old object entries. An object is stale if it has not been refreshed from the database within a configurable amount of time. Upon accessing a cache entry, the server calculates the duration since the object was last read from the database. If that duration exceeds the stale time, the server refreshes the cache entry from the database. To avoid increased complexity, PolicyCenter prefers this mechanism over evicting objects upon stale timeout. At runtime, you can use the **Cache Info** or **Management Beans** page to modify this value. See also: • "Cache Info" on page 142 • "Management Beans" on page 137
GlobalCacheStatsWindowMinutes	This parameter denotes a period of time, in minutes, that PolicyCenter uses for two purposes: • how long to preserve the reason that PolicyCenter evicted an object, after the event occurred. When a cache miss occurs, PolicyCenter reports the reason on the **Cache Info** page. • the period for which to display statistics on the chart on the **Cache Info** page. For more information on the **Cache Info** page, see "Cache Info" on page 142.
ScriptParametersRefreshIntervalSecs	The number of seconds between refreshes of the script parameters cache. This is a specialized cache only for script parameters. See "Special Caches for Rarely Changing Objects" on page 70.
ZoneCacheRefreshIntervalSecs	The number of seconds between refreshes of the zones cache. This is a specialized cache only for zones. See "Special Caches for Rarely Changing Objects" on page 70.

Analyzing Cache Settings

See "Cache Info" on page 142 for information on how to view cache performance. The cache performance information includes the number of objects currently in the cache, the number of objects evicted from the cache, and more.

The percentage of evictions is currently always set to 0. Cache hit ratio metrics are intrinsically dependent on the workflow that is using the object. Some workflows involve reading an object only one time while others involve reading the object many times. The cache hit varies depending on these workflows. There are therefore no good default cache hit ratios. Experimentation combined with performance measurements constitutes the only approach to identifying appropriate cache sizes. Also, if an application server started recently or has not had much user load, then hit rates can be skewed low. For example, if you recently started the server, and users have only visited a few pages, the hit rate is very low because PolicyCenter encountered only a few cache hits. As users visit more pages, the hit rate increases.

Detecting Cache Thrashing

You can find evidence of cache thrashing by:

1. Analyzing the number of evictions on the **Cache Info** page.
2. Resetting the **Cache Info** page.
3. Reproducing the operation.
4. Reanalyzing the **Cache Info** page.

If an individual cache reports hundreds or thousands of evictions and a low cache hit rate, then that cache is thrashing. If you notice cache thrashing on an application server node not processing batch jobs, resize the cache. Otherwise, dedicate the application server node to batch jobs.

After you have taken the proper action, repeat the analysis to ensure that the change yielded the expected results.

Special Caches for Rarely Changing Objects

In addition to the global cache, PolicyCenter includes caches specific to rarely changing objects. PolicyCenter includes a cache for each of the following:

- Exchange rates
- Groups
- Script parameters
- Zones

These caches periodically refresh the entire set of the rarely changing object. This prevents the application server from querying the database each time PolicyCenter accesses one of these objects, thereby improving performance. For each of these special caches, you can set the refresh interval. See “Analyzing and Tuning the Application Server Cache” on page 67.

Analyzing Server Memory Management

Java provides platform-side memory management that significantly simplifies coding. The JVM (Java Virtual Machine) periodically identifies unused objects and reclaims associated memory. This process is called garbage collection. Garbage collection can have a significant impact on application server performance.

This topic describes Java platform garbage collection analysis.

This topic includes:

Memory Usage Logging

The memory usage logging message looks like the following:

```
serverName 2007-04-09 16:44:14,423 INFO Memory usage: 80.250 MB used, 173.811 MB free, 254.062 MB
  total, 2048.000 MB max
```

- `used` – memory allocated to objects. This includes active objects which are still in use, and stale objects which will eventually be garbage collected.
- `free` – unallocated memory.
- `total` – amount of memory that the JVM process has reserved from the operating system.
- `max` – the maximum total memory that the JVM is allowed to use.

PolicyCenter writes this logging message if the parameter `MemoryUsageMonitorIntervalMins` in `config.xml` is set to a value other than the default of `0`.

It is common for the server to use up the maximum amount of memory fairly quickly, so that `used` and `total` are at or near the `max` value. This indicates normal operation. When the server needs more memory, it triggers garbage collection to free up the memory used by stale objects. The memory values printed in this logging line do not provide enough information to detect or analyze memory problems.

You only need to worry about memory issues if the server throws an `OutOfMemoryError` exception. If that happens, see the remaining sections in this topic to configure the garbage collector to print out detailed memory information.

This logging line cannot provide more detailed information, such as "used active" versus "used stale", without actually running the garbage collector. To do so just for the sake of more detailed logging would interfere with the optimal pattern of garbage collection and is not supported. Turn on garbage collector logging to get more detailed information on the memory usage of the application, as it is currently configured. See “Enabling Garbage Collection” on page 71.

The logging line is provided "as is" and is not configurable. The values provided by this logging line are not detailed enough to indicate or warn of memory issues. Only by turning on garbage collection logging can you get an accurate picture of memory usage.

Enabling Garbage Collection

The garbage collector can provide additional information on collection statistics. Careful analysis helps understand garbage collector behavior.

Enable verbose garbage collection by adding the `-verbose:gc` option to the Java Virtual Machine (JVM). Additional details can be found on the site of each JVM vendor.

IBM JVM

Enable verbose garbage collection by adding the `-verbose:gc` flag to the JVM options. Other options exist for the same functionality.

IBM estimates that the overhead associated with verbose garbage collection is minimal and estimated to be 2% of the garbage collection time. In other words, if the JVM spends 5% of its time garbage collecting without verbose garbage collection, it would spend 5.1% of the time garbage collecting with verbose garbage collection.

The output provided is in XML format. This output is generally rich enough for a thorough analysis, and no additional levels of logging are needed.

Used with WebSphere, the IBM JVM outputs garbage collection logs into a file called `native_stderr.log`.

IBM provides the IBM Support Assistant. You can install multiple plugins within this tool. Several plugins are available for the IBM JVM and WebSphere. These tools provide deep analysis of JVM behavior, spot issues, and recommend how to tune the JVM.

Oracle Java Hotspot VM

Enable verbose garbage collection by adding the `-verbose:gc` flag to the Java HotSpot VM options. Several levels of logging exist, providing more or less output.

The garbage collection time logs can time stamp the various entries with the exact date. Guidewire recommends the following options:

```
-XX:+PrintGCDetails -XX:+PrintGCDateStamps -XX:+PrintHeapAtGC
-XX:+PrintGCApplicationConcurrentTime -XX:+PrintGCApplicationStoppedTime
```

These options provide the following:

- Nature of the garbage collection (minor or full)
- Amount of memory reclaimed
- Time elapsed since JVM start or date corresponding to the event, depending on available options
- Before and after state of the different memory pools (nursery, tenured and permanent)
- Amount of time the application runs between collection pauses
- Duration of the collection pause

The level of information can be overwhelming, though it is necessary in some cases.

Add the `-Xloggc:file` option to redirect output to the specified file.

Analyzing Garbage Collector Behavior

Verify that the performance analysis tool you choose supports the version of the JVM that you use for PolicyCenter. For supported JVM versions, see the *Guidewire Platform Support Matrix*, available from the Guidewire Resource Portal at `http://guidewire.custhelp.com`.

IBM Support Assistant

IBM provides the IBM Support Assistant Workbench. Multiple plugin tools can be installed within the workbench. The "IBM Monitoring and Diagnostic Tools for Java – Garbage Collection and Memory Visualizer" is the tool to use to analyze garbage collection logs.

The tool provides some tuning recommendations. The recommendations are more adapted for the IBM JDK than the HotSpot JDK. Additionally, the tool provides graphs with hints on JVM behavior that help identify issues such as memory shortages or excessive pauses.

Refer to `http://www.ibm.com/software/support/isa/` for more information about the IBM Support Assistant.

HPjmeter and GCViewer

HPjmeter and GCViewer are tools that enable you to visually analyze the HotSpot JDK garbage collection logs. Both tools generate:

- Key metrics about the period (number of minor/major collections, percent of time spent paused, and so forth)
- Visual representation of the different garbage collections

These tools might require different verbose garbage collection options. Otherwise, HPjmeter or GCViewer might not be able to analyze the corresponding output.

Refer to the following URLs for more information:

HPjmeter – `http://h20000.www2.hp.com/bizsupport/TechSupport/Home.jsp`

GCViewer – `http://www.tagtraum.com/gcviewer.html`

Analyzing a Possible Memory Leak

Guidewire applications are memory-intensive. Guidewire applications generally require larger heaps than most other Java applications.

Garbage collection logs might show that memory usage grows significantly over time, resulting in a lack of available memory. This condition is commonly described as a memory leak. A dump of all used objects, called a "heap dump", must collected to identify all objects in the heap. This heap dump is then analyzed by developers familiar with PolicyCenter or additional code. Such analysis helps identify excessive memory usage, identify its root cause and possibly find a change that will avoid such issues.

Investigation of memory leaks differs slightly per JVM platform.

Common approach

Various options exist to generate a heap dump:

- Specific flags can be set to force the following behaviors:
  - Heap dump generation when the heap is full and an out-of-memory condition occurs
  - Heap dump generation when a `CTRL-BREAK` or SIGQUIT is issued to the JVM process.

These options are combined with options instructing the JVM to generate the heap dump at a specific directory location.

- Tools can connect to a running JVM. Such tools provide the option to trigger a heap dump.

Generating Heap Dumps with IBM JDK

The IBM JVM capabilities to generate heap dumps are satisfactory for all release levels that Guidewire has worked with. For information on generating heap dumps for IBM JDK 1.6, refer to "*IBM Developer Kit and Runtime Environment, Java Technology Edition, Version 6*" at `http://public.dhe.ibm.com/software/dw/jdk/diagnosis/diag60.pdf`.

Generating Heap Dumps with Oracle HotSpot JDK

Flags `HeapDumpOnOutOfMemoryError` and `HeapDumpOnCtrlBreak` can be used for heap dump generation.

For more information about analyzing heap dumps refer to Oracle document *Troubleshooting Guide for Java SE 6 with HotSpotVM* at `http://www.oracle.com/technetwork/java/javase/tsg-vm-149989.pdf`.

Additional Recommendations

When generating heap dumps, pay attention to the following facts:

- Heap dump generation frequently fails because the single file generated is very large and the supporting environment is configured to prevent regular accounts to create such large files. Therefore, some configuration is generally required to allow the account running the node to create such large files.
- The generation of a heap dump during out-of-memory conditions is sometimes challenging. As a JVM is reaching maximum memory utilization, it generally experiences severely degraded performance. As the pace of the leak decreases gradually, the occurrence of the out-of-memory condition might take an inordinate amount of time. This length of time might be incompatible with the need to restore performance for users or processes.
- Windows only: Windows does not support signals. Therefore, generating a heap by starting the JVM with a heap dump on `CTRL-BREAK`, depends on the capacity to send a `CTRL-BREAK`. You cannot send a `CTRL-BREAK` to a JVM started as a background process. Therefore, for the time of the investigation, start the JVM from a command line rather than as a background process.
- The JVM generally provides optional flags that prevent it from listening to signals. Disable these flags while trying to generate a heap dump through signals.
- Heap dump analysis is very memory intensive. Assume that the tool used to analyze the heap dump might need a heap two to three times larger than the amount of objects captured in the heap dump. Host the heap dump analyzer on a server with a 64-bit JVM and a significant amount of memory. If such a configuration is not available, you might want to reduce the heap size so that the memory leak reaches an identifiable threshold sooner. This method allows the generation of smaller, easier to analyze heap dumps.
- Heap dump analysis tools generally point to the `CacheImpl` class as the largest memory consumer. This class corresponds to the Guidewire cache. It is normal that the cache consumes a few hundred megabytes. In this case, the memory issue is likely not caused by cache growth. If the cache consumes significantly more memory, you might need to be downsize the cache. See “Application Server Caching” on page 65.

Heap Dump Generation and Analysis Tools

Several tools are available for heap dump generation and analysis. IBM and Oracle provide some tools to assist with these tasks on their respective JVMs. Some tools are provided by other vendors and aim to assist with these tasks on the most common JVM platforms.

IBM-only Tools

- IBM Support Assistant provides some plug-in tools that can assist with heap dump analysis. Refer to `http://www.ibm.com/software/support/isa/`.
- IBM DTJF adapter for Eclipse Memory Analyzer allows the Eclipse Memory Analyzer to analyze IBM JVM heap dumps. You can tune that tool to use a larger heap, which is frequently necessary to analyze very large heap dumps. Refer to `http://www.ibm.com/developerworks/java/jdk/tools/mat.html`.

Oracle-only Tools

- jConsole is a management tool that connects to a running Java HotSpot VM. You can trigger a heap dump by using jConsole. Refer to *Using JConsole to Monitor Applications* at `http://java.sun.com/developer/technicalArticles/J2SE/jconsole.html`.
- Oracle bundles the Java Heap Analysis Tool (jhat) with Java HotSpot VM 1.6. Therefore, if you want to analyze a heap with jhat, you can install Java HotSpot VM 1.6 and use the jhat release provided. Refer to the article *Java Heap Analysis Tool* at `http://docs.oracle.com/javase/6/docs/technotes/tools/share/jhat.html` for more information.
- jVisualVM is another multi-purpose tool that you can use to analyze heap dumps. Refer to *JVisualVM* at `http://docs.oracle.com/javase/6/docs/technotes/guides/visualvm/index.html`.

Generic tools

- YourKit is a commercial product that provides many functions. You can use YourKit to connect to the JVM, analyze the JVM and trigger heap dumps. It also provides some very interesting heap dump analysis tools.
- JProbe is another commercial product providing many capabilities, including heap dump analysis.

Guidewire development mainly uses YourKit with good success. Guidewire Support uses YourKit and several other products like jVisualVM, IBM DTJF adapter and JProbe.

Profiling

Java profilers are available for two main purposes:

- Memory profiling: profilers allow identifying memory usage and more specifically memory leaks due to referenced but unused objects.
- CPU profiling: profilers help identify programmatic hot spots/bottlenecks. This analysis might help remove the corresponding bottlenecks thereby increasing performance.

Guidewire has internally used two profiling tools that it found to be of good quality. Both tools provide both memory and CPU profiling:

- YourKit is preferred for memory profiling.
- JProfiler is preferred for CPU profiling.

To profile PolicyCenter, load the profiler agent into the PolicyCenter JVM either when starting PolicyCenter or by attaching the profiler agent to a running JVM. Refer to instructions for your profiler for instructions.

Tracking Large Objects

Large Java objects cause an extra strain on the JVM for various reasons. If garbage collection analysis shows that the JVM is allocating very large objects, investigate this further and understand the source of the objects.

chapter 6

Clustering Application Servers

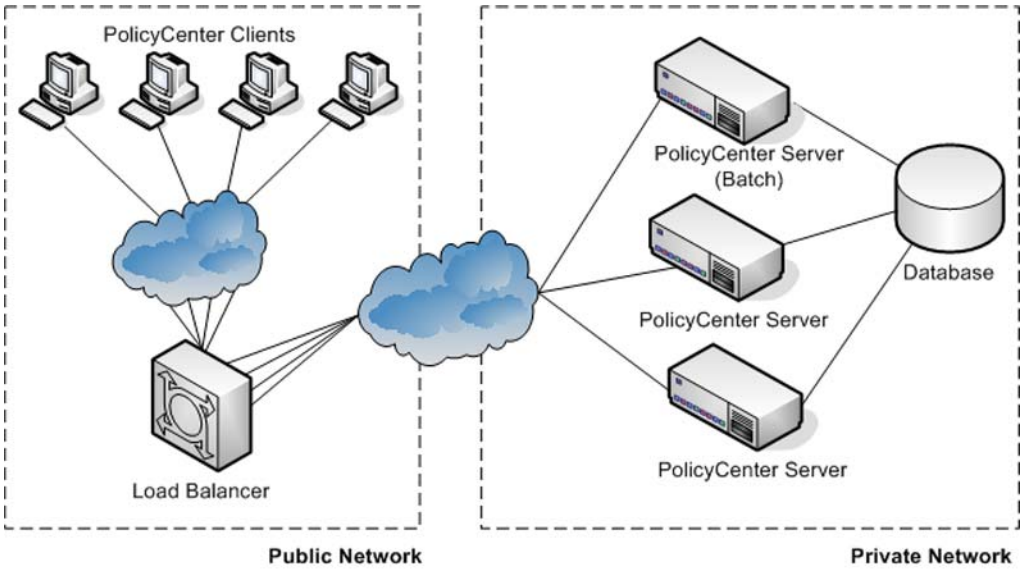
To improve performance and reliability, you can install multiple PolicyCenter servers in a configuration known as a cluster. A cluster distributes client connections among multiple PolicyCenter servers, reducing the load on any one server. If one server fails, the other servers seamlessly handle its traffic. This topic describes how to configure a PolicyCenter cluster.

Also review “Considerations for a Clustered Application Server Environment” on page 18 in the *Installation Guide*.

This topic includes:

Overview of Clustering

The typical clustered environment consists of multiple PolicyCenter servers, a single batch server, and a load balancer. The following diagram illustrates a clustered environment:



Plan the cluster so that if any one server fails, the other servers in the cluster can handle its traffic without being overwhelmed. PolicyCenter servers in the cluster can run on separate computers, or you can run multiple servers on the same computer. Guidewire recommends you maintain at least three PolicyCenter servers in the cluster, whether on the same or different physical computers. With multiple servers running on the same computer, in the event of a failure, then all servers are unusable. Of course, the exact configuration depends on specific usage needs.

To establish a cluster, you must also install your own load balancing solution. The load balancer acts as the bridge between client connections and the private network. Clients send a connection request to the load balancer and it routes the request to PolicyCenter server. The load balancer must implement *session affinity*, meaning that it must route connections from the same user session to the same PolicyCenter server. If the load balancer directed a user to a different server, the session is reset. This can result in loss of unsaved data.

Within any cluster, there can be only one PolicyCenter batch server. The batch server acts as a typical PolicyCenter server, and also performs system operations that would fail if multiple servers attempted to perform them. These operations include processes such as activity escalation and database upgrades. To ensure the batch server has adequate resources to run system processes, limit the traffic that the load balancer distributes to the batch server. Guidewire suggests that the batch server run on its own host computer. You can define multiple batch servers, so you can start another one if your primary batch server fails. However, do not start more than one batch server at the same time.

If you change script parameters, then shut down all non-batch servers before starting the batch server. Only the batch server writes script parameters from `ScriptParameters.xml` to the database. As the batch server starts, it retires script parameters and writes new values. If a non-batch server is using script parameters that the batch server changes during startup, non-batch servers can throw null pointer exceptions while trying to access the script parameters.

In general, start the batch server first. If it fails, either restart the batch server, start another batch server or use the Management Beans section of Server Tools to designate another node as the batch server. See “Management Beans” on page 137. If another server goes down that is not the batch server, you can restart that server without restarting each computer in the cluster.

Special Considerations Regarding PolicyCenter Batch Servers

PolicyCenter uses an application-side internal cache mechanism that limits database read attempts. This mechanism is critical in optimizing performance. During batch jobs, however, the batch server likely processes many objects, load-managing objects into the cache.

If this batch server is also being used to provide service to end users of Guidewire applications, both processes use the same caching mechanism. These two functions are likely to compete for caching resources, leading to a large number of cache evictions. Application users would then experience slower performance, as the server requires additional reads from the database.

Therefore, for installations with heavy or frequent batch processes, Guidewire recommends that no application users be served from the batch server. This is a not a strict requirement, and does not apply to all installations. Additionally, batch servers and application servers have different resource requirements, so it is unlikely that a full server would be dedicated to perform the role of batch server.

Finally, if the server has the processing resources necessary, this batch server can have other application instances alongside, provided that these different instances run within separate JVMs.

Notes:

- For security, Guidewire strongly recommends that the PolicyCenter servers and database reside within a protected private network, not directly accessible from outside sources.
- Enable necessary security measures to protect server side components such as application servers and databases.
- In a clustered system, it is theoretically possible to analyze inter-node communication and then generate unnecessary traffic with potential negative side effects amounting to denial of service attacks. Therefore, use network protection, such as enabling a DMZ with strict security rules.
- The administration tools and Guidewire Studio must connect directly to the PolicyCenter server. These tools cannot connect through a load balancing virtual host.
- Guidewire applications are application server independent. For this reason, if you implement a PolicyCenter cluster, Guidewire recommends that you not use proprietary application server features for failover, sharing sessions between nodes, and so forth. Instead, disable these features.
- You can designate multiple servers as capable of being the batch server. However, you can run only one batch server at a time.

Configuring a Cluster

You install PolicyCenter on all the servers in a cluster in the same way that you install a standalone PolicyCenter server. Then, you create the cluster by changing each server's configuration settings. With some exceptions, all the servers in a cluster must have identical `config.xml` files and identical metadata. There are directories within the `configuration` module (`PolicyCenter/modules/configuration`) that need not be identical among clustered servers, these directories are:

Directory	Contains
`config/import`	Files that you can only import manually. Guidewire recommends that you maintain these files on only one server.
`config/logging`	The logging configuration, which can be different for each server. For example, each server could log to a local directory, a shared file system, or a logging server.

As each server starts, it connects to another server in the cluster and compares its configuration environments. If the configurations differ, the server fails to start. To configure a cluster, you use several parameters in the `config.xml` file and make use of the PolicyCenter environment properties to ensure that server-specific properties resolve correctly. See the following sections:

- “Enabling and Disabling Clustering” on page 78
- “Configuring the Registry Element for Clustering” on page 78
- “Setting the Multicast Address” on page 80
- “Specifying the Key Range” on page 80
- “Configuring Separate Logging Environments” on page 80

Before continuing to configure the cluster, review “Defining the Application Server Environment” on page 14.

Enabling and Disabling Clustering

To enable clustering, set the `ClusteringEnabled` parameter in `config.xml` to `true` as follows:

```
<param name="ClusteringEnabled" value="true"/>
```

To disable clustering and remove a server from a cluster, set this parameter to `false`. After the server is no longer in a cluster, it behaves as any other standalone server.

Configuring the Registry Element for Clustering

The registry element and PolicyCenter environment properties play an important role in creating a cluster. Since the `config.xml` file and the `config` subdirectories must be identical, create a configuration that runs on all servers. If you have not already done so, review “Defining the Application Server Environment” on page 14. The following `registry` element illustrates a simple scenario for defining a cluster with environment properties:

```
<registry>
  <server serverid="buffy" isbatchserver="true" />
  <server serverid="spike"/>
  <server serverid="watcher"/>
</registry>
```

IMPORTANT Specify the `serverid` for the batch server by name rather than IP address. Batch processes might not run if the batch server is specified by IP address.

Of course, you need not use the registry element at all, you can use JVM options. See “Setting Java Virtual Machine (JVM) Options” on page 14 for more information.

Developing Sophisticated Configurations by Using Localized Parameters

Since all servers in a cluster must use an identical `config.xml` file, the value of most configuration parameters is the same for all of them. However, you can also develop more sophisticated configurations that make extensive use of the PolicyCenter environment properties and localized parameters. You can develop multiple configurations that work in multiple situations.

For example, you might develop a configuration that had the ability to run each server alone or in a cluster. The following configuration illustrates this.

```
<registry>
  <systemproperty name="env" value="my.env" default="cluster1"/>

  <server serverid="giles" env="cluster1" isbatchserver="true"/>
  <server serverid="spike" env="cluster1" isbatchserver="true"/>
  <server serverid="buffy" env="cluster1" isbatchserver="true"/>
</registry>
...
<param name="ClusteringEnabled" value="true" env="cluster1" />
<param name="ClusteringEnabled" value="false" env="standalone" />
...
```

At startup, PolicyCenter first resolves the `env` element. Assuming that `-Dgw.pc.env` is not set on the command line to another value, this `registry` would result in the `env` resolving to `cluster1`.

This same configuration can be used to start any of these servers in standalone mode. To do this, start the application server with a `-Dgw.pc.env=standalone` JVM option. Notice that, in the previous example, the localized parameter `ClusteringEnabled` resolves to a different value in the standalone environment than the clustered environment:

```
<param name="ClusteringEnabled" value="true" env="cluster1" />
<param name="ClusteringEnabled" value="false" env="standalone" />
```

For a complete discussion of localized parameters including a list of the parameters you can localize, see, “Specifying Parameters by Environment” on page 17.

Defining a Batch Server with the isbatchserver Environment Property

A batch server performs the following operations:

- Upgrades the database.
- Performs staging table operations.
- Dispatches integration messages.
- Starts scheduled processes, such as activity escalation and statistics calculation.

Only one PolicyCenter server can act as the cluster’s batch server. Having only one batch server prevents multiple servers from attempting to upgrade the same database and possibly running into conflicts over important resources. Since batch server operations can be resource intensive, a single batch server in a cluster reduces the network load.

Within the cluster, the `isBatchServer` property must resolve to `true` on at least one server. You can specify multiple servers as the batch server. This enables you to launch another server as the batch server if the current batch server stops working. However, only start one batch server at a time.

In a standalone configuration in which `ClusteringEnabled` is `false`, the individual server always acts as its own batch server.

Identify the batch server by setting the `isbatchserver` attribute of the `server` element in `config.xml` to `true`:

```
<registry>
  <server isbatchserver="true" serverid="hostname of batch server"/>
  <server serverid="hostname of the nonbatch server"/>
</registry>
```

The `serverid` is case-sensitive. Specify the `serverid` exactly as you named the server. Otherwise, PolicyCenter can not find the batch server.

Alternatively, you can edit the `config.xml` file and create a `systemproperty` of type `isbatchserver` to redefine the `gw.pc.isbatchserver` option:

```
<registy>
    <systemproperty name="isbatchserver" value="gw.cc.isbatchserver" default="false"/>
</registry>
```

For a complete discussion of how environment properties work, see “Defining the Application Server Environment” on page 14.

If you do not specify `isbatchserver` through the JVM options, PolicyCenter does the following to determine whether the server is a batch server:

1. PolicyCenter takes the `isbatchserver` value of the first `serverid` that it matches. Regardless of whether you set the `isbatchserver` parameter on the `server` subelement or not, the default value of `isbatchserver` is `false`. Therefore:

   ```
   <server env="cluster1" serverid="buffy"/>
   ```

   is the same as:

   ```
   <server env="cluster1" serverid="buffy" isbatchserver="false"/>
   ```

2. Checks for a default value defined in the `systemproperty` of type `isbatchserver`.

If none of the methods succeed, PolicyCenter sets `isbatchserver` to `false` by default.

WARNING You can start the servers in a cluster in any order, unless you have made data model changes. If you have made data model changes, start the batch server first.

Setting the Multicast Address

You must set the multicast address and port on each server in the cluster. Servers communicate with each other over this multicast address. Servers use multicast to communicate cache expiration messages and for control purposes, such as verifying configuration parameters and ensuring that only one batch server starts.

You specify the multicast address with the `ClusterMulticastAddress` and `ClusterMulticastPort` parameters in `config.xml`. For example:

```
<param name="ClusterMulticastAddress" value="228.9.9.9"/>
<param name="ClusterMulticastPort" value="45678"/>
```

The default values for these parameters are probably acceptable for a simple clustering arrangement. If you have multiple clustered environments, then the multicast address, and perhaps the port, must be different for each cluster as follows:

```
<param name="ClusterMulticastAddress" env="productioncluster" value="228.9.9.9"/>
<param name="ClusterMulticastPort" env="productioncluster" value="45678"/>

<param name="ClusterMulticastAddress" env="testcluster" value="228.9.9.10"/>
<param name="ClusterMulticastPort" env="testcluster" value="45679"/>
```

To be valid, a multicast address must be within the following specific range:

224.0.0.0 – 239.255.255.255

By convention, the Internet Assigned Numbers Authority (IANA) reserves certain addresses within the 224.0.x.x address space. See *IP4 Multicast Address Space Registry* at the following location for details:

```
http://www.iana.org/assignments/multicast-addresses/multicast-addresses.xml
```

Specifying the Key Range

When you create a new PolicyCenter object such as a Policy, PolicyCenter assigns it a key, or unique public identifier. To ensure that keys are unique, the server requests an available key from the PolicyCenter database. If every server in a cluster queried the database each time it needed a single key, performance would degrade. Instead, configure the servers to obtain a block of keys with a single request. For example, a server can reserve a block of 100 keys, and then assign them without needing to query the database again.

The server assigns keys from a block until the server uses all keys in the block. To set the number of keys the server obtains with each request, set the `KeyGeneratorRangeSize` parameter in `config.xml` as follows:

```
<param name="KeyGeneratorRangeSize" value="100"/>
```

This value is large enough to prevent frequent database queries for more keys, but small enough to not waste too many keys that the server reserves but never uses. The server discards allocated but unused keys when the server shuts down. Keys are 64-bit integers, so wasting a few keys is not an issue. The default value of 100 is reasonable in most situations.

Configuring Separate Logging Environments

You can use the `env` property to specify different logging files for different environments. To use this method, design the clustered environment with an `env` value that evaluates to something other than null, for example `cluster1`. Then, you create a `cluster1-logging.properties` file and place that file in the `PolicyCenter/modules/configuration/config/logging` directory on each server. If that file does not exist or the `env` property does not resolve to `cluster1`, PolicyCenter uses the default `logging.properties` file.

You can also specify unique logging files per server within the same environment. See “Configuring Logging in a Multiple Instance Environment” on page 24.

Managing a Cluster

This topic discusses the ongoing management of a clustered environment. This topic includes:

Starting Clustered Servers

To start PolicyCenter servers in a cluster

1. Start the batch server.
2. Wait until you see the following in the log for the batch server:

   ```
   date time INFO ***** PolicyCenter ready *****
   ```

3. Start web service PolicyCenter servers. After executing each server start, wait ten seconds. Then, start the next web service PolicyCenter server.
4. On each web service PolicyCenter server, check that the server has started by testing the server. Direct a browser to the PolicyCenter HTTP ping utility:

   ```
   http://server:port/pc/ping
   ```

   When the server is running at the default `MULTIUSER` run level, the browser displays the number `2`. For more information on the PolicyCenter HTTP ping utility, see “Checking Node Health” on page 81.
5. For each web service server, invoke `http://server:port/pc/soap/someWSAPI?WSDL` where *someWSAPI* is a web service that you have defined. This publishes all WSDLs and ensures that web services are ready.
6. Start regular PolicyCenter servers for handling user requests. After executing each server start, wait ten seconds. Then, start the next PolicyCenter server.

   The last step assumes that the PolicyCenter servers for handling user requests make web services calls to web service servers. If this is not the case, then you do not need to wait for complete startup of web service servers before starting the regular PolicyCenter servers. Instead, just wait for ten seconds after the previous server startup script is executed.

Checking Node Health

Typically, hardware or software load balancers check the health of the various nodes and stop directing traffic to a node that stops responding. This check is very summary and is limited to verifying that the corresponding network port responds. Therefore, it is possible that a load balancer redirects traffic to a node that is not capable of processing that traffic appropriately. Some examples are that PolicyCenter is:

- Not fully started yet.
- At the `MAINTENANCE` run level.
- Experiencing significant issues, such as an out of memory condition.

Guidewire applications include a simple HTTP ping utility that enables you to check the application status with a web browser. For instance, to check the status of an instance of PolicyCenter running on port 8080 of the local computer, you would enter the following URL into a web browser:

```
http://localhost:8080/pc/ping
```

At least three possible responses can be discerned from a web browser:

- If the application is running at the default `MULTIUSER` run level, the browser displays the number `2`.
- If, however, the application is running in any mode other than `MULTIUSER`, the browser might display a non-display character.
- If the application server is not running, the browser displays an HTTP failure message, depending on the configuration of the server.

Guidewire based this HTTP ping utility on a system run level checker built for developers. See “Getting and Setting the Run Level” on page 97 in the *Integration Guide*. Invoking this utility programmatically provides more granular information on the server’s status.

Load balancers can be configured to regularly access this URL and determine the health of each node in the cluster. These results can be used to create redirection logic.

Adding a Server to a Cluster

Before you add a server to a cluster, create a backup of both the configuration of the server that you plan to add and the cluster. If you have been maintaining your configurations under source control, this might not be necessary. Regardless of whether you use source control or manual backup, a backup enables you to return to the original configuration if something goes wrong.

Within a cluster, not only the `config.xml` file and all `config` subdirectories must match among all computers in the cluster. If you add a server to a cluster, you might want to identify how that server differs from the cluster configuration. If there are differences, decide whether the server ignores the differences or updates the base cluster configuration.

To add a server to the cluster

1. On the server you want to add, remove the `config` directory and the `config.xml` file.
2. Copy the `config` directory and the `config.xml` file from a server on the cluster to the server you want to add.
3. If the new server is on the same physical machine as another server, set the `serverid` system property in the `config.xml` file to a unique value within the cluster. By default, the `serverid` defaults to the hostname of the machine running the instance. This works well in most configurations but causes problems if multiple instances are running on a single machine. See “Defining the Application Server Environment” on page 14.
4. Start the new server.

When you start the new server, it connects to the cluster and compares its configuration with the cluster configuration. It performs a checksum of the `config.xml` file and checks the `config` subdirectories. If the configurations differ, the server fails startup. PolicyCenter writes failure messages to the log file:

```
-------------------------------------------------------
GMS: address is havasu:3491
-------------------------------------------------------
st:8085/pc  2006-05-05 14:16:02,963 INFO Cluster channel started
st:8085/pc  2006-05-05 14:16:02,963 INFO Starting clustered inittab
st:8085/pc  2006-05-05 14:16:02,963 INFO Started clustered inittab
st:8085/pc  2006-05-05 14:16:02,978 INFO Starting clustered config verifier
st:8085/pc  2006-05-05 14:16:13,979 ERROR Local server configuration doesn't match configuration for
  servecdcdr havasu:3476
st:8085/pc  2006-05-05 14:16:13,979 ERROR
  File config\elements\test.txt was found on the remote server, but not on the local server
st:8085/pc  2006-05-05 14:16:13,994 ERROR
  An exception was thrown while starting a component. Setting runlevel to
  SHUTDOWN [Configuration mismatches]
com.guidewire.GWLifecycleException: Configuration mismatches at
  com.guidewire.pl.system.cluster.ClusteringComponent.start(ClusteringComponent.java:153)
```

Checking Server Run Level

You can check on the health of a particular node in the cluster through an unauthenticated web page. This page is located at a URL of the following format:

```
http://server:port/pc/ping
```

If the node is listening, this page returns a code indicating the server run level.

Code	Run level
2	MULTIUSER
-	DAEMONS
(	MAINTENANCE

Server Failures and Removing a Server

Although a cluster can reduce the impact of a server failure, there is no automatic procedure for detecting a failed server and restarting it. If a server in a cluster fails, you must restart the server manually to have it rejoin the cluster.

To remove a server from the cluster

1. Shut the server down.
2. Change the server configuration to a standalone configuration.
3. Deploy the new configuration.
4. Restart the server.

You can also create a configuration that supports a secondary batch server if the primary fails. For example, you can a define a cluster like this:

```
<registry>
  <systemproperty name="env" value="my.env" default="cluster1"/>

  <server serverid="giles" env="cluster1" isbatchserver="true"/>
  <server serverid="spike" env="cluster1" isbatchserver="true"/>
  <server serverid="buffy" env="cluster1" isbatchserver="true"/>
</registry>
```

With this configuration, only start one batch server. Only one server can function as the batch server at a time. However, in the event that one of the servers fails, another server is already designated as a possible batch server. You do not need to reconfigure and redeploy a new configuration across the cluster. Instead, simply start another server with `isbatchserver` set to `true`.

If you stop a batch server, then the operations for which it is responsible no longer run. Scheduled batch processes do not run. In addition, integration messages continue to queue up, but are not sent to their destinations. Start another batch server if you stop the current batch server.

Running Administrative Commands

Although the servers are clustered, server management is not. You can not set a particular server mode or start a batch process on all the servers with a single command. Instead, you must run the command individually on each server in the cluster.

Updating Clustered Servers

To update a server, shut it down, deploy an updated application file, and start the server again. For more information, see “Deploying PolicyCenter to the Application Server” on page 82 in the *Installation Guide*.

When you restart an upgraded PolicyCenter server, PolicyCenter might need to upgrade the database. Therefore, when restarting servers in a cluster, start the batch server first, and wait for it to complete the database upgrade before starting other servers.

WARNING You can start the servers in a cluster in any order, unless you have made data model changes. If you have made data model changes, start the batch server first.

chapter 7

Securing PolicyCenter Communications

Guidewire products use a standard three-tier architecture:

- The browser tier presents the PolicyCenter interface to the user.
- The web/application tier processes business logic.
- The database tier stores data.

Encryption secures communication between computer systems. You can secure the communication between the browser, web server and application server to a level strong enough that it cannot be easily compromised. This section provides an overview of what is required to secure these communications.

IMPORTANT Computer security and encryption is a complex topic in which network architecture plays a major role. Use this documentation as a starting point. Guidewire strongly recommends that you also do independent research and testing to develop a secure solution for your company network and installed applications. Guidewire strongly recommends that you deploy PolicyCenter over SSL (secure socket layer) for at least the login and change password pages. Ideally, deploy PolicyCenter entirely under SSL to protect all sensitive transmitted data.

This topic includes:

Using SSL with PolicyCenter

A strong password policy is the first and best line of defense. Consider encrypting communication between the Internet and the application server. Consider configuring a separate server to act as an intermediary layer between the Internet and application server. Typically, this intermediary server is located in a “DMZ” you establish through your network architecture.

You can use a web server or proxy both to encrypt communications and to provide a layer between the Internet and application server. Using a server as an intermediary in this manner is called a reverse proxy. If you offload encryption to a server, be aware that non-native encryption processing is not as efficient. Native applications generally use optimized encryption modules. The example in this documentation uses encryption provided by a native application.

There are multiple methods you can use to achieve an encrypted proxy solution. The example in this document creates a reverse proxy that interacts with the PolicyCenter application server through HTTP. This is similar to how a regular user accesses the server.

Overview of the Steps

This example uses the Apache HTTP server as the reverse proxy server. To set up the proxy server, you must first download and install the Apache HTTP server software appropriate to your environment (for example, `httpd-2.2.22-win32-x86-openssl-0.9.8t.msi`). Follow the directions supplied by the Apache documentation to install the CRT and PEM certificate. Then, follow the procedures in the following sections:

1. “Editing the httpd.conf File” on page 86
2. “Editing the httpd-ssl.conf File” on page 86
3. “Editing the server.xml File” on page 87

Editing the httpd.conf File

You configure the Apache server by using directives placed in plain-text configuration files. On start up, the server locates and reads the `/Apache2/conf/httpd.conf` file within the Apache installation directory. Modify this file to load the following modules:

`mod_proxy`	Enables proxying.
`mod_proxy_http`	Enables HTTP proxying.
`mod_ssl`	Enables SSL tunneling.
`mod_deflate`	Compresses output from the server.

Edit the `httpd.conf` file, look for and uncomment the following lines.

```
LoadModule proxy_module modules/mod_proxy.so
LoadModule proxy_http_module modules/mod_proxy_http.so
LoadModule ssl_module modules/mod_ssl.so
LoadModule deflate_module modules/mod_deflate.so
Include conf/extra/httpd-ssl.conf
```

Editing the httpd-ssl.conf File

The `/Apache2/conf/extra/httpd-ssl.conf` file within the Apache installation directory defines configuration settings for the SSL module.

To edit the Apache SSL module

1. Add SSL listening port numbers for each port number. The default provided in the file is:

   ```
   Listen 443
   ```

2. For every new listening port add a virtual host context.

   ```
   #Encrypted Reverse Proxy
   <VirtualHost *:portnumber>

         #Allow from the authorized remote sites only
         <Proxy *>
               Order Deny,Allow
               Allow from all
         </Proxy>
   ```

```
      # Access to the root directory of the application server is not allowed
      <Directory />
            Order Deny,Allow
            Deny from all
      </Directory>

      #Access is allowed to the pc8.0.3 and
      #its subdirectories for the authorized sites only
      <Directory /pc8.0.3>
            Order Deny,Allow
            Allow from all

            # Never allow communications to be not encrypted
            SSLRequireSSL

            #The Cipher strength should be 128 (maximal cipher size authorized
            #all communication will be secured
            SSLRequire %{SSL_CIPHER_USEKEYSIZE} >= 128 and %{HTTPS} eq "true"
      </Directory>

      #Classic command to take into account an Internet Explorer issue
      SetEnvIf User-Agent ".*MSIE.*" \
      nokeepalive ssl-unclean-shutdown \
      downgrade-1.0 force-response-1.0

      #Encryption secures the Internet to Encrypted Reverse Proxy communication
      #Listing of available encryption levels available to Apache
      SSLEngine           on
      SSLCipherSuite      ALL:!ADH:!EXPORT56:RC4+RSA:+HIGH:+MEDIUM:+LOW:+SSLv2:+EXP:+eNULL

      #The Virtual Host authenticates to the user providing its certificate
      SSLCertificateFile      conf/<certificate_filename>.crt
#The communication security is achieved using the PrivateKey, which is secured through
      #a pass-phrase script.
      SSLCertificateKeyFile   conf/<certificate_filename>-secured.pem
#The Virtual Host associates the request to the internal Guidewire product instance
      ProxyPass           /pc http://MyPolicyCenterHost:8080/pc
      ProxyPassReverse    /pc http://MyPolicyCenterHost:8080/pc
#Logs redirected to appropriate location
      ErrorLog      logs/encrypted_<product>.log
</VirtualHost>
```

3. Save and close the file.
4. Install Apache as a service with SSL support:

   APACHE_INSTALL_DIR/Apache2/bin/httpd -D SSL -k install

5. Start Apache.

   ```
   net start Apache2
   ```

At this point, you have configured the reverse proxy. Users make requests through their browser to this server. The reverse proxy server encrypts the requests and forwards them to the application server.

Editing the server.xml File

When PolicyCenter responds to a request, it requires the URL and port that originated the request. By default, this location is directly accessible to users. When you add a reverse proxy, as in this example, that proxy lies between the user making the request and the PolicyCenter application server. To support the reverse proxy server, you must edit the application server configuration so it is aware of:

- the externally-visible domain name of the reverse proxy server
- the port number of the reverse proxy server
- the protocol the client used to access the proxy server (in this case HTTPS)

This example assumes you are using the Apache Tomcat application server. To make the server aware of the proxy, edit the `CATALINA_HOME/conf/server.xml` on the deployment server and add an additional connector:

```
<!-- Define a non-SSL HTTP/1.1 Connector on port <port number>
     to receive decrypted communicated communication from Apache
     reverse proxy on port 11410 -->
<Connector acceptCount="100"
  connectionTimeout="20000"
```

```
    disableUploadTimeout="true"
    enableLookups="false"
    maxHttpHeaderSize="8192"
    maxSpareThreads="75"
    maxThreads="150"
    minSpareThreads="25"
    port="portnumber"
    redirectPort="8443"
    scheme="https"
    proxyName="hostname"
    proxyPort="portnumber">
</Connector>
```

Specify the following:

`port`	Specifies the port number for the additional connector for access through the proxy. For example, `8080`.
`proxyName`	Identifies the deployment server's name. For example, `MyApacheHost`.
`proxyPort`	Specifies the port for encrypted access through Apache. For example, `443`.
`scheme`	Identifies the protocol used by the client to access the server.

After configuring the `server.xml` file, restart the application server.

Accessing a PolicyCenter Server Through SSL

Use a different address to access PolicyCenter through SSL. The new address resembles the following:

```
https://server:port/pc/PolicyCenter.do
```

Notice the use of the `https` protocol instead of `http`, indicating that the server is connecting through a secure version of HTTP. In addition, use the new port number on which the proxy server is running.

This address must change in every client that connects to the server, including web browsers, PolicyCenter plugins, and applications that use PolicyCenter APIs. Also, check `config.xml` for any URL specifications that you need to change.

Handling Browser Security Warnings

When users connect to PolicyCenter over the SSL connection, they might see a security warning stating that they are connecting to a secure server. Users can click **Yes** to proceed and connect to PolicyCenter. The next time users connect in a new browser session, the warning appears again.

To disable this warning

1. Open Internet Explorer.
2. Select **Tools** → **Internet Options**.
3. Choose the **Security** tab.
4. Select **Web Content Zone** as either **Internet** or **Intranet**.
5. Press the **Custom Level** button.

   Internet Explorer displays the **Security Settings** dialog.
6. Scroll to **Miscellaneous Setting** section.
7. Change the **Display Mixed Content Setting** radio button from **Prompt** to **Enabled**.
8. Click **OK** to close the dialog and save your change.
9. Click **OK** to close the **Internet Options** dialog.

chapter 8

Importing and Exporting Administrative Data

This topic discusses the key concepts and procedures you need to know about importing administrative data from an external system into PolicyCenter. While users enter much of the information into PolicyCenter directly, at times it is more convenient or necessary to enter information in bulk. This topic also discusses how to export data.

This topic includes:

Understanding Data Import and Export

PolicyCenter needs to maintain information about how you organize people and their work. In many cases, you can edit this information within the PolicyCenter administration screens. However, you might want to import it from a file, in the same way PolicyCenter loads sample data. Information you can import includes:

- **Users and groups** – the people and organization descriptions in your company.
- **Contacts** – vendors, companies, and other business entities.
- **Role definitions** – the list of roles and the permissions granted to each role.
- **Security zones** – the list of separate zones that limit access to information for people who are not members of the zone.

- **Activity patterns** – templates that standardize the way PolicyCenter generates activities. Both Gosu classes and the user interface create activities based on these patterns. Each pattern describes one kind of activity that might be needed in handling the policy or account process. Activity patterns contain many default, or typical, characteristics for each activity, such as its name, its relative priority, and whether it is mandatory. When either you or a Gosu class create an activity, PolicyCenter uses the pattern as a template to set the activity's default values, such as Subject and Priority. Defaults can be overridden.
- **FNOL** – first notice of loss data

You must edit and load these files while first setting up PolicyCenter.

The next two sections describe the process and tools PolicyCenter offers for importing and exporting data.

What Mechanisms are Available to Import and Export Data?

You can use the following mechanisms to import and export administrative data:

Method	Description
CSV files	Create one or more CSV files and upload them by using the `import_tools` command. This method is useful for development environments in which multiple people share sample data. For a detailed explanation see “Importing Administrative Data from the Command Line” on page 92.
user interface	Create one or more import files containing administrative data and import them using the PolicyCenter **Administration** tab. You must have the `viewadmin` and `soapadmin` permissions to access this option. This method also warns against and enables you to resolve collisions between incoming data and existing data. You can also use the user interface to export administrative data. For a detailed explanation, see “Importing and Exporting Administrative Data from PolicyCenter” on page 95. You can also import sample data for use in development. See “Installing Sample Data” on page 55 in the *Installation Guide*.
staging tables (zone data only)	You can use the functionality provided by the `zone_import` command or the `ZoneImportAPI` web service to import zone data to staging tables. See “Zone Import” on page 541 in the *Integration Guide* for detailed information about this mechanism. Also see “Zone Import Command” on page 169. Staging table import is only available for zone data.
Gosu	You can create one or more Gosu entity builder classes that define data.

The PolicyCenter Data Model

To import or export data from PolicyCenter, you must understand its data model. In particular, understand how PolicyCenter uses each user interface field and how that use maps to the database. To build this understanding, review the *PolicyCenter Data Dictionary*, located at:

```
PolicyCenter/build/dictionary/data/index.html
```

If you do not see the `dictionary` directory, run the following command from `PolicyCenter/bin`:

```
gwpc regen-dictionary
```

The *Data Dictionary* describes the structure of business objects stored persistently by PolicyCenter, and the dictionary defines the properties and foreign key references for each object. If you change the data model, regenerate the *PolicyCenter Data Dictionary* by using the command shown above.

See also

- “Working with the Data Dictionary” on page 143 in the *Configuration Guide*

Public ID Prefix

Each entity that you import into PolicyCenter requires a unique public ID. This is separate from the system ID that PolicyCenter assigns internally and uses for most system processing. Foreign key references between related objects use this public ID. Later, if the external system needs to locate the data it previously imported, it can make use of this public ID. PolicyCenter can locate the data and update it correctly.

For example, an external system might have an account (`A1`) with a specific policy (`P1`). The import data must be structured such that PolicyCenter knows that policy `P1` is associated with account `A1`. Additionally, the external system might need to locate or even update objects in a PolicyCenter database after the initial import. For example, suppose a policy `P2` is added to the account `A1`.

Typically, a company imports data from multiple external sources. If you do this, use a naming convention to generate public IDs for external sources. For example, if you import from two systems (`Adminsystem` and `Salessystem`), each could have a contact entity with ID=5432. By using the following format, you can ensure that the IDs do not register as duplicates:

```
origin:ID
```

By using this format, the contact from the first system comes in as `adminsystem:5432` and the contact from the second system comes in as `salessystem:5432`. So, there is no risk of duplicate IDs. You also have the benefit of knowing from which system the record originated.

Note: The IDs need to be unique only within objects of the same type. For example, all policies must have a different public ID. However, an account and a policy with the same public ID do not conflict. Public IDs cannot exceed 20 characters in length.

Support for Unique Public IDs in a Development Environment

During development and testing of a PolicyCenter configuration, multiple developers can create administrative data in their own PolicyCenter installations. Administrative data includes users, groups, roles and so forth. You combine this data into a single production database at the end of the development cycle.

If you will ever transfer data from one database to another with the export and import utilities, the two databases must have different public ID prefixes. To ensure that the administrative data from one system does not overwrite another as they are combined, have each developer set a unique public ID prefix. The public ID prefix is set by modifying the `value` of the following parameter in `config.xml`:

```
<param name="PublicIDPrefix" value="pc"/>
```

PolicyCenter appends this public ID prefix to each administrative entity created in an individual PolicyCenter configuration. If a developer exports data from that installation, each public ID has a format of:

```
{PublicIDPrefix}:{ID}
```

Coordinate among engineers to ensure that no public ID prefixes overlap. For example, engineers might use their initials or computer names as a public id prefix.

You can use the same prefix for multiple development and testing databases if you do not ever transfer data between them.

What Happens During an Import?

As PolicyCenter loads external data from the command line, it uses the public ID to determine whether it already knows about the object. If PolicyCenter finds a match, it reinserts the existing record instead of adding a new record. When PolicyCenter reinserts the data, it overwrites any existing values. If there are differences, the incoming values overwrite the current values in the PolicyCenter database.

WARNING If you use `import_tools` to update existing records in PolicyCenter, send the entire object, not just the property to be changed. PolicyCenter assumes that any missing properties are empty and removes any information that was previously there.

You can import administrative data using the PolicyCenter **Administration** page instead of the command line. You must have the `viewadmin` and `soapadmin` permissions to access this feature. PolicyCenter enables you to step through differences between imported data and data in the database. At this point, you can choose to accept the imported data or keep what is in the database.

Importing Administrative Data from the Command Line

Guidewire provides an import tool for loading object data into the PolicyCenter system. PolicyCenter reads import data from a `CSV` (comma-separated values) file. To import data, create a `CSV` file describing the data, either manually or by exporting from another application such as Microsoft Excel. Then, import that file with the `import_tools` command as follows:

```
import_tools -password password -import fileName
```

The `import_tools` command is located in the `PolicyCenter/admin/bin` directory.

The `PolicyCenter/modules/configuration/config/import/gen` directory contains samples of the various data types that you might want to import.

IMPORTANT The `MaximumFileUploadSize` parameter in `config.xml` must exceed the size of any file from which you want to import. The `MaximumFileUploadSize` parameter value is in megabytes (MB). The default value of `MaximumFileUploadSize` is 20 MB.

Creating a CSV File for Import

You can only import data for an entity that already exists in the Guidewire data model. Each file you import must have a heading that defines what the file contains and how to interpret it. The following example illustrates a very simple import file:

```
1: ADDRESS
2: type,data-set,entityid,addresstype,addressline1,createuser
3: Address,0,ab:1001,home,1253 Paloma Ave.,import_tools
4: Address,0,ab:1002,business,325 S. Lake Ave.,import_tools
```

The `import_tools` command distinguishes between two types of information in an import file: heading information and data information. The command treats any line that contains the string `entityid` as a heading. The command considers as data any line:

- Without an `entityid` string
- Containing comma delimited values
- Containing a value in its third comma-delimited field

In the previous example, the `import_tools` command treats line 2 as a heading and lines 3-4 as data. The `import_tools` command ignores line 1. If the command encounters a data line before a heading line, it returns an error.

Constructing a Heading

A heading line initializes the import for a particular entity object in the data model. A heading consists of comma-separated fields. The first three fields must be, in order, `type`, `data-set`, and `entityid`. Subsequent fields must refer to columns, typelists, foreign keys, and joinarrays on the entity.

For example, a file importing into the `Address` entity has a heading that appears as:

```
type,data-set,entityid,addresstype,addressline1,createuser
```

The three required fields are followed by the `addresstype` field, which represents a typelist, and the `addressline1` field, which represents a column. You do not have to specify all fields in the entity within the import file. You must specify at least the required fields. You can determine which fields PolicyCenter requires by viewing the entity description in the *PolicyCenter Data Dictionary.*

Use lowercase to specify fields, including arrays. In this example, specify `AddressLine1` in the data model as `addressline1` in the import file.

To specify a foreign key, use the foreign key name without the concluding ID. In this example of a `Person` import:

```
1: type,data-set,entityid,firstname,lastname,primaryaddress,
  workphone,primaryphone,taxid,vendortype,specialtytype
2: Person,0,demo_sample:1,Ray,Newton,demo_sample:4000,818-446-1206,work,,,
3: Person,0,demo_sample:2,Stan,Newton,demo_sample:4002,818-446-1206,work,,,
4: Person,0,demo_sample:3,Harry,Shapiro,demo_sample:1004,818-252-2546,work,,,
5: Person,0,demo_sample:4,Bo,Simpson,demo_sample:1003,619-275-2346,work,,,
6: Person,0,demo_sample:5,Jane,Collins,demo_sample:4003,213-457-6378,work,,,
7: Person,0,demo_sample:6,John,Dempsey,demo_sample:1006,213-475-9465,work,,,
```

The `primaryaddress` field is a foreign key to the `Address` entity. It appears as `PrimaryAddressID` in the *PolicyCenter Data Dictionary* but as `primaryaddress` in the import data.

If you specify a field in the heading that is not a recognizable column, typelist, foreign key or array, the import program silently ignores the column and any associated data. In the following example, the import ignores the `%$zed` heading field and the `somedata` value in line 3:

```
1: ADDRESS
2: type,data-set,entityid,addresstype,addressline1,createuser, %$zed
3: Address,0,ab:1001,home,1253 Paloma Ave.,import_tools, somedata
4: Address,0,ab:1002,business,325 S. Lake Ave.,import_tools,
```

Working with Data for Import

The `import_tool` identifies data lines by looking for lines (without the three required heading fields) that contain comma-delimited values and whose third field is non-empty. Each data line represents a single instance of a data model entity. The first field in any data line must be an entity name or an entity subtype name.

```
1: Policy
2: type,data-set,entityid,account,corepolicynumber,policytype,
  producttype,productversion,systemofrecorddate,,,,,,,,,,
3: Policy,0,ds:1,ds:1,34-123436-CORE,wc,wc_workerscomp,1,1/1/2002,,,,,,,,,,
4: Policy,0,ds:2,ds:1,25-123436-CORE,bop,bop_businessowners,1,1/1/2002,,,,,,,,,,
5: Policy,0,ds:3,ds:3,54-123456-CORE,personalauto,pa_personalauto,1,1/1/2002,,,,,,,,,,
6: Policy,0,ds:4,ds:4,25-708090-CORE,bop,bop_businessowners,1,1/1/2002,,,,,,,,,,
7: Policy,0,ds:5,ds:2,98-456789-CORE,bop,bop_businessowners,1,1/1/2002,,,,,,,,,,
8: Policy,0,ds:6,ds:1,20-123436-CORE,businessauto,ba_businessauto,1,1/1/2002,,,,,,,,,,
9: Policy,0,ds:7,ds:1,50-123436-CORE,umbrella,u_umbrella,1,1/1/2002,,,,,,,,,,
```

In lines 3 - 9, the entity name `Policy` appears in the first field as required. The capitalization of an entity or subtype name must be identical to that used in the *PolicyCenter Data Dictionary.* For example, to create a `RevisionAnswer` data line the entry name would be invalid if you specified it as `revisionanswer`.

The second field in a data line is the value of the highest-numbered data-set of which the imported object is part. Data-sets are ordered by inclusion, thus data-set 0 is a subset of data-set 1 and data-set 1 is a subset of data-set 2, and so forth. It is possible to request a particular data-set while converting CSV to XML. By default, the data-set of 10240 is requested. It is assumed that data-set 10240 includes every data-set that might be created in practice. The second field can be left blank, in which case PolicyCenter always includes this object in the import regardless of which data-set is requested.

The third field in any data line must be the entity's public ID. This field is mandatory. For example, `ds:2` is the public ID of the `Policy` on line 4.

Foreign Key and Column Data

The `import_tools` command imports both column and typelist data values from the CSV file. In the previous example, the `policytype` column has a value of `wc` in line 3 and a value of `bop` in line 4. You represent foreign key data by a string in one of two formats:

publicID

or

entity_id:identity_source

If there is more than one : (colon), the import ignores everything after the second : (colon).

```
1  ADDRESS
2  type,data-set,entityid,addresstype,addressline1,createuser
3  Address,0,ab:1001,home,1253 Paloma Ave.,import_tools
4  Address,0,ab:1002,business,325 S. Lake Ave.,import_tools
6
7  PERSON
8  type,data-set,entityid,firstname,lastname,primaryaddress,
   inaddressbook,loadrelatedcontacts,
    preferred,contactaddresses
9  Person,0,ab:2001,John,Foo,ab:1002,true,true,true,
   ContactAddress|address[ab:10001,ab:1002]
10 Person,0,ab:2002,Paul,Bar,ab:1002,false,false,false,
   ContactAddress|address[ab:10001]
11 Person,0,ab:2003,David,Goo,,false,true,false,,
```

In the previous example, the primaryaddress on line 9 is a foreign key to the Address specified on line 4.

If PolicyCenter cannot resolve a foreign key reference and the foreign key is not required, PolicyCenter imports the data, setting the foreign key field to null, and reports an error. If the foreign key is required, then PolicyCenter reports an error and does not import that data.

Simple Array Data

You specify simple array data, referencing a single foreign key, by using the following format:

arraykey|*foreignkey*[*publicID*,*publicID*,...]

In the PERSON example (line 9), the *arraykey* value is the array key on the parent entity (Person). The foreignkey is the foreign key name of the array without the ID. ContactAddress is the array key and address is the foreign key name. The public ID values [*publicID*,*publicID*,...] correspond to public IDs reference by the foreign key.

In this format, the arraykey is optional. However, you might want to retain it for readability.

Complex Array Data

You might need to specify more complex arrays that have a mixture of data types. If you specify arrays that contain a mixture of columns, foreign key data, or typelists, use a different format. The basic format of these complex array entries appear as follow:

[[*array_entry*];[*array_entry*]; ...]

Enclose each *array_entry* in brackets. Separate multiple entries with semicolons. Enclose all completed entries in a second set of brackets. Each array_entry is made up of comma-separated [*type*|*value*] pairs as follows:

[[[*type*|*value*],[*type*|*value*]];[[*type*|*value*],[*type*|*value*]]]

The *type* is the name of a column, typelist, or foreign key, as in a heading line. The *value* is the column value, typelist typecode, or a foreign key. In the following sample, there are three *array_entry* specifications, the first and last *array_entry* specifications appear in bold:

```
Group
type,data-set,entityid,users
Group,0,demo_sample:27,[[user|demo_sample:101],
  [loadfactor|50],[loadfactortype|loadfactorview]];[[user|demo_sample:102],
  [loadfactor|100],[loadfactortype|loadfactoradmin]];
  [[user|demo_sample:103],[loadfactor|50],[loadfactortype|loadfactorview]]]
```

Using CSV Files with Different Character Sets

A CSV file that you create might contain characters not recognized by the default character encoding. For example, you might have a CSV file with accented characters, or with umlauts. To use other character encodings, set the import tool's `-charset` option: If the unusual (accented) characters are encoded as single bytes, `-charset ISO-8859-1` might be appropriate. If they are double bytes, you might specify `-charset UTF-8`.

Maintaining Data Integrity While Importing

Some PolicyCenter administrative data has dependencies on business roles. For example, roles are associated with groups. Therefore, if you export administrative data from one system into another, then you must also export and import the roles. Perform these steps in the following order.

To maintain data integrity while importing data

1. Export roles.
2. Export administration data.
3. Import roles into the new system.
4. Import administration data into the new system.

Importing and Exporting Administrative Data from PolicyCenter

You can import and export administrative data from the PolicyCenter user interface rather than from the command line. This interface imports and exports data in XML format only.

Creating an XML File for Import

The `PolicyCenter/build/xsd/pc_import.xsd` file defines the XML schema used for import and export. This file further references information in two other XSD files in the same directory: `pc_entities.xsd` and `pc_typelists.xsd`. You can use any schema-aware XML editor to help format information properly according to these XSD definitions. You generate these XSD files with the following command:

```
gwpc regen-xsd
```

Regenerate the XSD files any time you modify the data model. These files are likely to change as you configure the data model.

Within an XML file, it is common to have references between objects in the file. For example, a user object might refer to a group of which it is a member. Since the group definition is elsewhere in the XML file, or perhaps was previously defined elsewhere, the user definition refers to this group with a foreign key. The foreign key is the object's public ID. For example, the XML file could contain:

```
<User publicID="demo_sample:100"> … </User>
…
<Group publicID="demo_sample:200">
  …
  <Users>
    <GroupUser>
    <User publicID="demo_sample:100" />
    …
  </GroupUser>
  </Users>
</Group>
```

In this example, the user `demo_sample:100` is a member of group `demo_sample:200`.

Within a single XML file you can reference an item before defining the item. This enables you to define all of the groups first, for example, including referring to supervisor users who are not defined until later in the file. PolicyCenter reports errors only if a referenced object still does not exist after reading the entire file.

Validating the XML

You can validate the XML of your import file against an `pc_import.xsd` file using the following code:

```
Uses java.io.File
Uses javax.xml.validation.SchemaFactory
Uses javax.xml.XMLConstants
Uses javax.xml.parsers.SAXParserFactory
Uses java.io.ByteArrayInputStream
Uses org.xml.sax.HandlerBase

var schemaFile = new File(TestEnvironment.TempDirectory, "pc_import.xsd")
assertTrue(schemaFile + " Should exists", schemaFile.exists());
var factory = SchemaFactory.newInstance(XMLConstants.W3C_XML_SCHEMA_NS_URI);
var schema = factory.newSchema(schemaFile)

var spf = SAXParserFactory.newInstance();
spf.setSchema(schema);
spf.Validating = true
spf.NamespaceAware = true
var parser = spf.newSAXParser();

parser.parse(new FileInputStream("myImportFile.xml"), new HandlerBase());
```

Importing Data From the User Interface

You can import administrative data from the user interface.

While importing data, PolicyCenter looks for matching data by `publicID` only. PolicyCenter notifies you if it locates existing records that match the data you are trying to import. You can then choose how you want to resolve differences between the existing data and the data you are importing.

If you perform a case-by-case resolution of each discrepancy, PolicyCenter displays conflicting fields and enables you to update the record with the new version or keep the old version.

During import, PolicyCenter inserts a new entity or updates an existing entity for each entity in the XML file, just as with the command-line import.

To import administrative data from PolicyCenter

1. Log on as a user with the `viewadmin` and `soapadmin` permissions.
2. Click the **Administration** tab.
3. Click **Utilities** → **Import Data**.
4. Click **Browse...** to search for the XML file containing data to import.
5. Click **Finish** to import data from the file.

During an import, PolicyCenter does not run validation rules. However PolicyCenter does run pre-update rules. For this reason, run user exception and group exception batch processing after you import administrative data.

Importing Arrays

PolicyCenter handles arrays differently depending on whether it is importing an owned array or a virtual array. If an entity owns the array, PolicyCenter notifies you of differences between the imported data and any existing data. However, you do not have the choice of resolving the array elements. PolicyCenter only gives you the option to delete the current array and replace all of the contents of the array. Virtual arrays, because they cannot be deleted, cannot be replaced by an import at all.

Exporting Data from the User Interface

You can export different sets of data from PolicyCenter.

You can export or import the following different sets of data: **Admin**, **Policy Forms**, **Policy Holds**, and **Roles**.

During export or import of users and groups, PolicyCenter also exports or imports any entities referred to by any `User` or `UserRole` object through a foreign key or array.

The **Export** command exports an XML file.

To export administrative data from PolicyCenter

1. Log on as a user with the `viewadmin` and `soapadmin` permissions
2. Click the **Administration** tab.
3. Click **Utilities** → **Export Data**.
4. Select the data set to export.
5. Click **Export** to download the XML file.

Other Import Functions

This section introduces other import functions available with PolicyCenter.

Importing a Table from an External Database

Guidewire supplies a `table_import` command for populating PolicyCenter tables with data from an external database. The `PolicyCenter/admin/bin` directory contains the `table_import` command. The `table_import` command acts on populated staging tables. After you populate the staging tables, use `table_import` to:

- Check the integrity of data in the staging tables
- Populate an exclusion table
- Delete rows from the staging table based on an exclusion table
- Run an integrity check and then load the data into PolicyCenter

IMPORTANT You can only import zone data by using staging tables.

Guidewire provides the `TableImportAPI` web service that performs all of the same functions as the `table_import` command. This API enables you to develop integrations that can do regular imports of zone data from external databases. See *"Zone Import" on page 541 in the Integration Guide* for detailed information about this API and converting zone data from external sources.

The `table_import` command requires that you set the server to the maintenance run level by using the `-maintenance` flag with the `system_tools` command. See "Using the Maintenance Run Level" on page 61.

The import Directory

When you install PolicyCenter, the database upgrader loads default roles and privileges. After the install, you can import XML files containing administrative data to add to or change these values. You must have the `viewadmin` and `soapadmin` permissions to access this feature.

You can modify files to bypass the user interface if you need to make large scale changes. Unless otherwise noted, these files can be accessed from the Guidewire Studio **Resources** pane under **configuration** → **config** → **import** → **gen**. Make changes in the files in Studio and reimport the files with the `import_tools` command as follows:

```
import_tools -password password -import fileName
```

The `import_tools` command is located in the `PolicyCenter/admin/bin` directory.

You must use the command line to import CSV files. The user interface only supports importing XML files.

Configuring Roles and Privileges

The file that defines roles and their names is `roles.csv`. The file `roleprivileges.csv` contains the mappings that link roles to a set of permissions. These files both use the same file format as the import data. For example, a `roles.csv` contains entries such as the following:

```
Roles,,,,,
type,data-set,entityid,description,name,carrierinternalrole
Role,0,adjuster,Base permissions for an adjuster,Adjuster,true
Role,0,claims_supervisor,Base permissions for a claims supervisor,Claims Supervisor,true
Role,0,manager,Base permissions for a manager,Manager,true
...
Role,0,newloss_supervisor,Base permissions for a claims supervisor,New Loss Processing Supervisor,true
Role,0,reporting_admin,Permissions for reporting admin,Reporting Admin,true
,,,,,
```

The `name` and `description` fields can be whatever helps you remember for whom you intended this role.

Then, the set of permissions granted to each role appear in the `roleprivileges.csv` file. One row maps a single permission to a role. Each role has multiple permissions and so multiple rows:

```
type,data-set,entityid,permission,role
RolePrivilege,0,default_data:1,abedit,adjuster
RolePrivilege,0,default_data:2,abview,adjuster
RolePrivilege,0,default_data:3,actcreate,adjuster
...
RolePrivilege,0,default_data:108,viewsnapshot,adjuster
RolePrivilege,0,default_data:109,viewvacation,adjuster
RolePrivilege,0,default_data:110,viewworkplan,adjuster
RolePrivilege,0,default_data:111,catmanage,catastrophe_admin
RolePrivilege,0,default_data:112,catview,catastrophe_admin
RolePrivilege,0,default_data:113,abcreate,claims_supervisor
...
```

For example, the `abview` entry grants permission to view the address book to the `adjuster` role. A full list of permissions, along with a brief description of each, is in the *PolicyCenter Data Dictionary.* See the `SystemPermissionType` typelist. See the *PolicyCenter Security Dictionary* for a list of the correspondence between roles and permissions.

chapter 9

Batch Processing

PolicyCenter provides tools for configuring and managing various forms of batch processing. You can schedule batch processing to run regularly or on demand.

This topic includes:

See also

- "Custom Batch Processing" on page 567 in the *Integration Guide*.

Overview of Batch Processing

PolicyCenter supports two styles of batch processing:

- **Work queue** – A work queue operates on a batch of items in parallel. Work queues run partially on the active batch server and can run on other servers in a PolicyCenter clustered environment.

  A work queue comprises the following components:

  - A processing thread, known as a *writer*, that selects a batch of items to process
  - A queue of selected work items
  - Processing threads, known as *workers*, that process the items to completion

  Work queues are suitable for high volume batch processing that requires items to be processed in parallel to achieve an acceptable throughput rate.

- **Batch process** – A batch process operates on a batch of items sequentially. Batch processes run only on the active batch server in a PolicyCenter clustered environment.

  Batch processes are suitable for low volume batch processing that achieves an acceptable throughput rate when it processes items sequentially. For example, writers for work queues operate as batch processes because they can select items for a batch and write them to their work queues relatively quickly.

Work Queues

Work queues run partially on the batch server. However, most of the batch processing work of a work queue is distributed in parallel to a number of servers in a cluster.

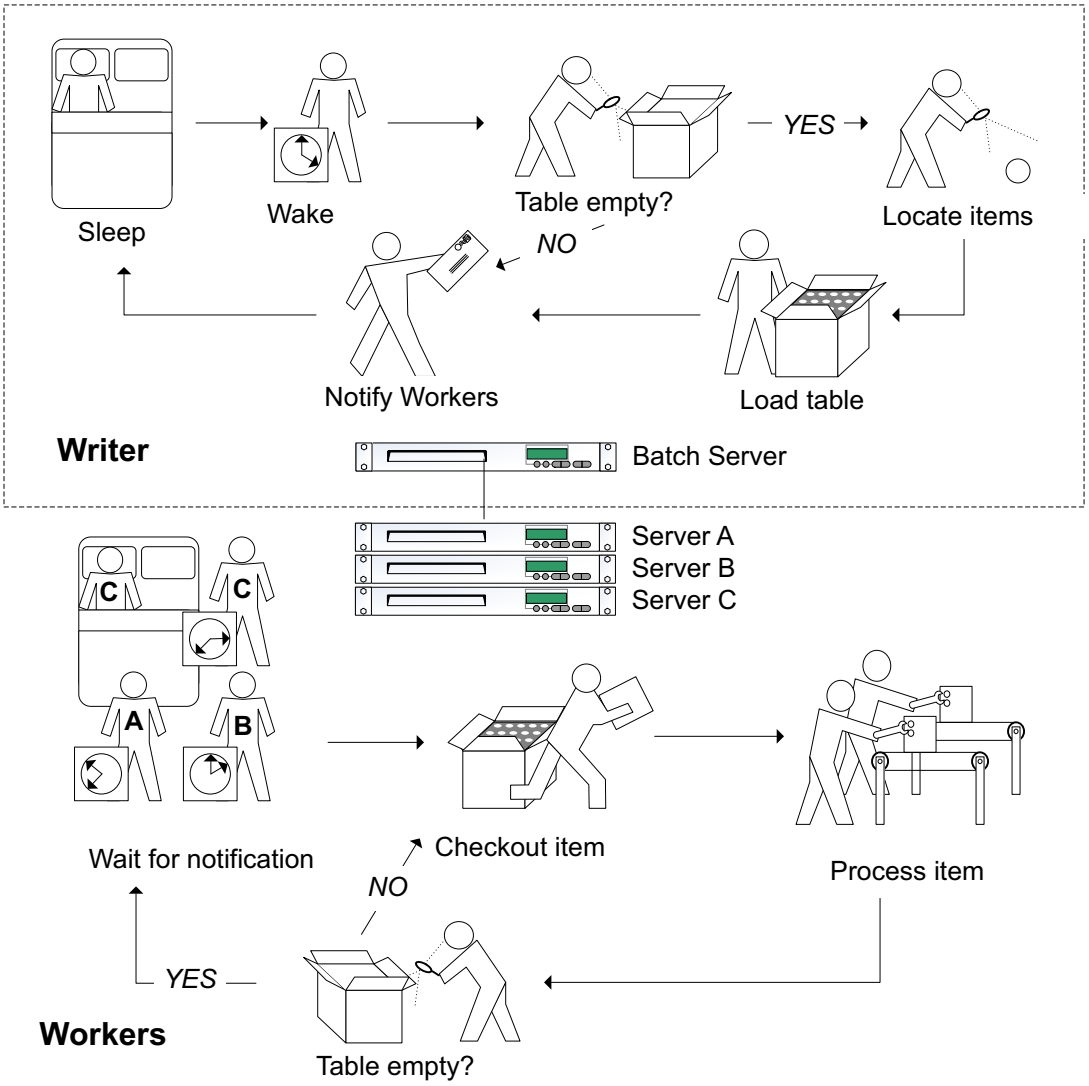
A work queue comprises the following components:

- **Writer** – A *writer thread* selects units of work for batch processing and writes their identities to a work queue. Writers are schedulable batch processes that run only on the batch server.
- **Work queue** – A work queue is a database table that the writer loads with a batch of work items and from which workers check out work items for processing.
- **Worker** – One or more *worker threads* check out work items from the work queue and process them to completion. Worker threads can run on any servers in a cluster, including the batch server.

Starting the writer initiates a run of batch processing on a work queue. The batch is complete when the workers exhaust the queue of all work items, except those that they failed to process successfully.

Work Queue Architecture

The following diagram illustrates the components of a work queue and how they function.



Functions of the Writer

Whenever the writer thread awakes or starts on demand, it checks the work queue table for any work items that remain from a prior batch.

If work items remain from a previous batch, the writer thread:

1. Notifies the workers that they have work to process.
2. The writer thread ends.

If no work items are found, the writer thread:

1. Selects items for a new batch.
2. Writes the identifies of the selected items to the work queue table.
3. Notifies the workers that they have work to process.
4. The writer thread ends.

Functions of a Worker

Typically, work queues share the standard work item table for their work items. However, a worker thread operates only on work items in that table inserted by its associated writer. For example, the Activity Escalation work queue might be configured for six workers on three different servers in a PolicyCenter cluster. Those workers work only on activity escalation items in the standard work item table. Typically, you configure work queues for multiple worker threads, so typically some number worker threads are operating throughout the day on items in the standard work queue table.

Whenever a worker thread awakens, it checks the work item table for work items from its associated writer. Sometimes workers are notified of work but find none available when they awaken. For small batches, a worker thread can check out all the items in the batch with its first quota between the time writer notifies the workers and other workers awaken. If a worker awakens and finds no work items, the worker goes back to sleep.

If a worker awakens and finds work items available for processing, the worker checks out its quota from the work queue. For each item, the worker sets the following attributes.

`Status`	Set to `checkedout`. This attribute can be `available`, `checkedout`, or `failed`.
`LastUpdateTime`	Set to the time when the worker checks out the work item.
`CheckedOutBy`	Set to the worker.

After it checks out a quota of work items, the worker thread processes them sequentially. Whenever a worker completes a work item successfully, it deletes the item from the table and begins to process the next item. The standard work item table (`StandardWorkQueue`) is retireable, so successfully completed work items remain in the table for historical reference.

Work Queue Scheduling and Processing Intervals

A writer for a work queue starts at the interval specified in `scheduler-config.xml`. Typically, you schedule the writer to start several times during the day or once at night. The writer thread runs on the batch server, just like a batch process. Access the schedule configuration file `schedule-config.xml` in the **Project** window of Guidewire Studio at **configuration** → **config** → **scheduler.**

Worker threads awaken much more frequently than their writers start. One worker awakens at least every `maxpollinterval` if not more frequently. You do not schedule worker threads. Instead, they awaken in response to notification from the writer or upon expiry of the polling interval. After a worker awakens, if there are work items to process, it processes up to `batchsize` items. If there are more items than the batch size to process, the worker awakens another worker. This worker repeats the process of checking out work items and waking up another worker if necessary, until the configured number of workers is reached. You configure the number of workers, polling interval, and batch size in `work-queue.xml`. Access the work queue configuration file `work-queue.xml` in the **Project** window of Guidewire Studio at **configuration** → **config** → **workqueue.**

See also

- “Scheduling Work Queues and Batch Processes” on page 107
- “Configuring Work Queues” on page 105
- “Performing Custom Actions After Batch Processing Completion” on page 110

Batch Processes

Batch processes execute on the batch server only. Generally, a batch process runs to completion and then reports its result back to a log or to the administrative user interface. You can view and manage batch processing from the **Batch Process** page on the **Server Tools** tab in the PolicyCenter application.

See also

- “Running Batch Processes” on page 103

Running Batch Processes

You can run many batch processes, including writers for work queues, from PolicyCenter or from the command line.

See also

- "Running Batch Processes Using Web Services" on page 96 in the *Integration Guide*

Running a Batch Process from PolicyCenter

You can run many batch processes, including writer batch processes for work queues, from the **Batch Process Info** screen in PolicyCenter. The **Batch Process Info** screen also contains information such as the current status of the batch process, when it last ran, when it will run again, and the schedule.

Note: You can run writers for work queues from the **Work Queue info** page or the **Batch Process Info** page.

Users must have the `internaltools` permission to access the **Batch Process Info** page. The Admin role has this permission by default. Alternatively, if the `EnableInternalDebugTools` parameter is set to true in `config.xml` and the server is running in development mode, all users can access to the **Batch Process Info** page.

To run a batch process from the Batch Process Info page

1. Log in to PolicyCenter.
2. Press `ALT+SHIFT+T` to display the **Server Tools** tab.
3. Click **Batch Process Info** in the left sidebar.
4. Click **Run** in the **Action** column of the batch process that you want to run. The **Run** button is enabled for all batch process types that belong to the `BatchProcessTypeUsage` category `UIRunnable`.

See also

- "Batch Process Info" on page 134
- "Server Modes and Run Levels" on page 58

Running a Writer from PolicyCenter

You can run the writer for a batch process from either the **Batch Process Info** screen or the **Work Queue Info** screen. From the **Work Queue Info** screen, you run the writer and then see the progress of workers processing the items in the batch. The process history that you download from the **Work Queue Info** screen includes history for the run of the writer, including its duration and starting and ending times.

Users must have the `internaltools` permission to access the **Work Queue Info** page. The Admin role has this permission by default. Alternatively, if the `EnableInternalDebugTools` parameter is set to true in `config.xml` and the server is running in development mode, all users can access to the **Work Queue Info** page.

To run the writer for a work queue from the Work Queue Info page

1. Log in to PolicyCenter.
2. Press `ALT+SHIFT+T` to display the **Server Tools** tab.
3. Click **Work Queue Info** in the left sidebar.
4. Click **Run Writer** in the **Action** column of the work queue. The **Run Writer** button is enabled for all work queue types that belong to the `BatchProcessTypeUsage` category `UIRunnable`.

See also

- “Work Queue Info” on page 135
- “Server Modes and Run Levels” on page 58

Running a Batch Process from the Command Line

You can run many batch processes, including writers for work queues, by running the `maintenance_tools -startprocess` command from the command line.

To run a batch process from the command line

1. Start the PolicyCenter server if it is not already running.
2. Open a command window.
3. Navigate to `PolicyCenter/admin/bin`.
4. Run the following command:
   ```
   maintenance_tools -password password -startprocess process
   ```
   For the *process* value, specify a valid process code.

See also

- For a list of process codes for batch processes, including writers for work queues, see “List of Work Queues and Batch Processes” on page 113.

Terminating a Batch Process from the Command Line

You can terminate in-progress processes, including writers for work queues, by using the `maintenance_tools -terminateprocess` command. However, *single phase* batch processing cannot be terminated. Single phase processes run in a single transaction, so there is no convenient place to terminate the process.

The following processes are single phase processes that cannot be terminated:

- `DataDistribution`

To terminate a batch process from the command line

1. Open a command window.
2. Navigate to `PolicyCenter/admin/bin`.
3. Run the following command:
   ```
   maintenance_tools -password password -terminateprocess process
   ```
   For the *process* value, specify a valid process code or a process ID.

See also

- For a list of batch process codes, including writers for work queues, see “List of Work Queues and Batch Processes” on page 113.

Checking Status of a Batch Process from the Command Line

You can check the status of processes, including writer processes for work queues, by using the `maintenance_tools -processstatus` command.

To check the status of a batch process from the command line

1. Open a command window.
2. Navigate to `PolicyCenter/admin/bin`.

3. Run the following command:

```
maintenance_tools -password password -processstatus process
```

For the *process* value, specify a valid process code or a process ID.

For work queues, this command returns the status of the writer process. It does not check whether there are remaining work items. It is possible for the process status to report as completed because the writer has completed adding items to the work queue, yet there are remaining unprocessed work items.

See also

- For a list of batch process codes, including work queue writer processes, see "List of Work Queues and Batch Processes" on page 113.

Configuring Work Queues

To configure a work queue, you configure attributes of the work queue and its workers. In addition, you can schedule the writer to run regularly, just as you would a batch process.

- Configure the work queue and worker attributes in the `work-queue.xml` file. Access this file in Guidewire Studio at **configuration** → **config** → **workqueue**.
- Configure the writer to run on a schedule in the `scheduler-config.xml` file. Access this file in Guidewire Studio at **configuration** → **config** → **scheduler**. See "Scheduling Work Queues and Batch Processes" on page 107.

After you edit either `scheduler-config.xml` or `work-queue.xml`, you must rebuild and redeploy PolicyCenter.

Each work queue has its own configuration structure in `work-queue.xml`.

```
<work-queue workQueueClass="string" progressinterval="decimal">
  <worker instances="integer" throttleinterval="decimal" env="string" server="string"/>
  ...
  <worker instances="integer" throttleinterval="decimal" env="string" server="string"/>
</work-queue>
```

The `<work-queue>` element has the following attributes:

Attribute	Description
`workQueueClass`	**Required.** The `workQueueClass` must be one of the Guidewire-provided work queue classes listed in `work-queue.xml`. These work queue classes implement the `BatchProcess` interface. You cannot configure custom batch processes as work queues. You also cannot configure Guidewire-provided batch processes as work queues if they are not already defined as work queues in `work-queue.xml`.
`progressinterval`	**Required.** The `progressinterval` value is the amount of time, in milliseconds, that PolicyCenter allots for a worker to process `batchsize` work items. If the time a worker has held a batch of items exceeds the `progressinterval`, then PolicyCenter considers the work items to be orphans. PolicyCenter reassigns orphaned work items to a new worker instance. The `progressinterval` must be greater than the time to process the slowest work item, or that work item will never be completed. Also, Guidewire recommends that you set `progressinterval` greater than the processing time of the whole batch. If a batch takes more time than `progressinterval`, PolicyCenter checks the remaining work items back in to the database. If there are many batches that take longer than `progressinterval`, the repeated checking out and checking in of work items can impact performance negatively.
`retryInterval`	**Optional.** How long in milliseconds to wait before retrying a work item that threw an exception. The default value is `0`, meaning PolicyCenter retries processing the item immediately.

Attribute	Description
retryLimit	**Optional.** How many times PolicyCenter retries a work item that threw an exception or became an orphan for this work queue. If you do not specify a retryLimit value for a work queue, PolicyCenter uses the value of the WorkItemRetryLimit configuration parameter in config.xml as the default value.
logRetryableCDCEsAtDebugLevel	**Optional.** If logRetryableCDCEsAtDebugLevel is set to true for a work queue, PolicyCenter logs any retryable Concurrent Data Change Exception (CDCE) at the DEBUG level. The log message includes a prepended string indicating that the error is considered to be non-fatal. Any CDCE that pushes the retry count over the retryLimit or workItemRetryLimit (if retryLimit is not set) is logged at the ERROR level.

For each work queue, you can declare as many worker instances as you want, specifying on which server and environment each one runs. If you do not specify a worker for a queue, or you do not specify the env and server attributes, PolicyCenter starts a single instance on the batch server. See "Specifying Environment Properties in the <registry> Element" on page 15 for information about the env and server attributes.

The <worker> subelements take the following attributes, all of which are optional:

Attribute	Description
instances	The number of workers to create. By default, PolicyCenter always creates at least one worker. Guidewire recommends an upper limit of 100 instances per server.
maxpollinterval	How often a worker will wake up automatically and query for work items, even if the worker receives no notification. If a worker wakes up and detects work items, it will check out those work items. If there are more work items than the batchsize, the worker will start another worker. Each worker will check out up to batchsize work items and then start another worker if there are more work items remaining until the number of instances is reached. You might need to increase maxpollinterval to prevent excessive numbers of queries for work items. The default maxpollinterval is 60,000 milliseconds.
throttleinterval	The delay between processing work items in milliseconds. The value controls how long the process sleeps. A value of 0 (zero) means worker threads process work items as rapidly as possible. To reduce the CPU load, set the throttleinterval to a positive value.
batchsize	How many work items the worker attempts to check out while searching for more work items. Larger batch sizes are more efficient, but might not result in good load distribution. The default batchsize is 10.
env	The environment in which this particular worker is active.
server	The server on which this particular worker is active.

See also

- To configure work queues dynamically by using web services, see "Manipulating Work Queues Using Web Services" on page 96 in the *Integration Guide*.

Worker Thread Management

An *executor* manages the worker threads on each server. One executor is created per work queue per server. The executor creates a single worker at server start up, regardless how many workers are configured for that server. The lone worker then checks the work queue for work items periodically on the configured frequency. If it finds work items, the executor creates additional worker threads, one by one, until all configured workers are active or the work queue is exhausted of non-failed work items. After the work queue is exhausted, the executor shuts down all workers except one.

Scheduling Work Queues and Batch Processes

The PolicyCenter scheduler launches many batch processes, including writer processes for work queues, according to a schedule defined in `scheduler-config.xml`. Access this file in Guidewire Studio at **configuration** → **config** → **scheduler**. The `scheduler-config.xml` file contains entries in the following format:

```
<ProcessSchedule process="process_code" env="environment">
  <CronSchedule schedule_attributes/>
</ProcessSchedule>
```

The *`process_code`* is the process to run. The *`environment`* is an optional attribute that specifies the environment in which the schedule definition for the process applies. The *`schedule_attributes`* is a valid schedule specification. See "Defining a Schedule Specification" on page 107.

If needed, you can list multiple `ProcessSchedule` entries for the same process. The process then runs according to each specified schedule. If you schedule a process to run while the same process is already running, then PolicyCenter skips the overlapping process. If the `scheduler-config.xml` file does not list a process, then the process does not run.

Generally, schedule the amount of time between batch process runs in hours as opposed to minutes. This is because some batch processes require a lot of resources on a server. Schedule such processes to wake infrequently or at times that the server is less taxed, such as late at night or very early in the morning.

The PolicyCenter scheduler uses the application server time for reference.

This topic includes:

Determining if a Batch Process Can Be Scheduled

Many batch processes, including work queue writers, can be scheduled. However, not all batch processes can be scheduled.

To determine if a batch process can be scheduled

1. Log in to PolicyCenter as a user with `internaltools` and `toolsBatchProcessview` permissions.
2. Press `ALT+SHIFT+T` to access the **Server Tools**.
3. Select **Batch Process Info** if not already selected.
4. Select **Schedulable** from the drop-down. PolicyCenter displays only those batch processes, including work queue writers, that can be scheduled in `scheduler-config.xml`.

Defining a Schedule Specification

The `CronSchedule` element describes when the process is run. It contains *`schedule_attributes`* that specify the exact timing, such as one time an hour or every night. The *`schedule_attributes`* is a combination of one or more of the following attributes:

Attribute	Standard Values	Default	Example
`seconds`	`0-59`	`0`	`seconds="0"`

minutes	0-59	0	minutes="15"
hours	0-23	*	hours="12"
dayofmonth	1-31	*	dayofmonth="1"
month	1-12 or JAN-DEC	*	month="2"
dayofweek	1-7 or SUN-SAT	?	dayofweek="1"

Along with the standard values listed, there are some special characters that give you more flexible options.

Character	Description
*	All values. For example, `minutes="*"` means run the process every minute.
?	Used to mean no specific value. Used only for `dayofmonth` and `dayofweek` attributes. See the examples for clarification.
-	Specifies ranges. For example, `hour="6-8"` specifies the hours 6, 7 and 8.
,	Separates additional values. For example, `dayofweek="MON,WED,FRI"` means every Monday, Wednesday, and Friday.
/	Specifies increments. For example, `minutes="0/15"` means start at minute 0 and run every 15 minutes.
L	The last day. Used only for `dayofmonth` and `dayofweek` attributes. See the examples for clarification.
W	Used for `dayofmonth` to specify the nearest weekday. For example, if you specify `1W` for `dayofmonth`, and that day is a Saturday, the trigger will fire on Monday the 3rd. You can combine this with `L` to schedule a process for the last weekday of the month by specifying `dayofmonth="LW"`.
#	used to specify "the nth" day of the week within a month. For example, a `dayofweek` value of 4#2 means "the second Wednesday of the month" (day `4` = Wednesday and `#2` = the second one in the month).

These represent only some of the values that you can use for setting a schedule. The PolicyCenter scheduler is based on the open source Quartz Enterprise Job Scheduler, and therefore uses the same specification for schedule attributes that Quartz uses. To determine the exact Quartz version, check the filename of the Quartz JAR file in `PolicyCenter/admin/lib`.

The following examples show some common ways to use the `CronSchedule` element. For additional examples, refer to the Quartz documentation. See `http://quartz-scheduler.org/documentation/quartz-2.1.x/tutorials/crontrigger` for more details and examples.

Example	Description
`<CronSchedule hours="10" />`	Run every day at 10 a.m.
`<CronSchedule hours="0" />`	Run every night at midnight.
`<CronSchedule minutes="15,45" />`	Run at 15 and 45 minutes after every hour.
`<CronSchedule minutes="0/5" />`	Run every five minutes.
`<CronSchedule hours="0" dayofmonth="1" />`	Run at midnight on the first day of the month.
`<CronSchedule hours="12" dayofweek="MON-FRI" dayofmonth="?" />`	Run at noon every weekday (without regard to the day of the month).
`<CronSchedule hours="22" dayofmonth="L" />`	Run at 10 p.m. on the last day of every month.
`<CronSchedule hours="22" dayofmonth="L-2" />`	Run at 10 p.m. on the second-to-last day of every month.
`<CronSchedule minutes="3" hours="8-18/2" dayofweek="1-5" dayofmonth="?"/>`	Run 3 minutes after every other hour, 8:03 a.m. to 6:03 p.m., Monday through Friday.
`<CronSchedule minutes="*/15" hours="0-8,18-23"/>`	Run every 15 minutes after the hour, 12:15 a.m. to 8:45 a.m. and 6:15 p.m. to 11:45 p.m.
`<CronSchedule hours="0" dayofmonth="6L" />`	Run at midnight on the last Friday of the month.
`<CronSchedule hours="4" dayofmonth="4#2" />`	Run at 4 a.m. on the second Wednesday of the month.

Determining the Current Schedule

To determine the current schedule of batch processes, you can either inspect the `scheduler-config.xml` file, or you can use the **Batch Process Info** page in PolicyCenter.

To determine the schedule in PolicyCenter

1. Log in to PolicyCenter with an administrative account.
2. Press `ALT` + `SHIFT` + `T` to navigate to the **Server Tools** tab.
3. Click **Batch Process Info**.
4. Click the **Next Scheduled Run** column header to sort processes by schedule. If a process is not scheduled, the **Next Scheduled Run** field is blank.

Scheduling Batch Processes Sequentially to Avoid Problems

Guidewire batch processes run best if you schedule them to run sequentially so that only one batch runs at a time. If batch processes run concurrently, throughput performance can degrade significantly and a high rate of concurrent data change exceptions can occur.

For example, two separate batch processes each run on average for half an hour. If you run the two sequentially, the total time from start to finish is about an hour. You might decide to shorten the processing window by running the two batch processes concurrently. However, running the two concurrently can actually lengthen the processing window rather than shortening it.

Batch processes that run concurrently share a common cache. The cache demands of each process end up flushing the cache more frequently, so fewer cache hits occur for each process. That increases the amount of physical reads from the relational database, thus degrading performance. In addition, concurrent data change exceptions can occur when each batch process attempts to update the same cached entity instances. This requires one or the other batch process to retry an item, leading to further performance degradation.

You can use the internal PolicyCenter scheduler to schedule batch processes far enough apart that they do not overlap. Alternatively, you can use your own scheduler to ensure that one batch process finishes before the next process begins.

Using the PolicyCenter Scheduler to Ensure Batch Processes Do not Overlap

The internal PolicyCenter scheduler does not let you specify that one batch process must finish before another one begins. The PolicyCenter scheduler is purely a time-based scheduler. If you use the PolicyCenter scheduler, schedule the processes in your chain of nightly batch processes far enough apart so they are unlikely to overlap.

Using Your Own Scheduler to Ensure Batch Processes Do Not Overlap

Your organization may have its own scheduler for starting batch processes. If so, you can verify that the work of one batch process is complete before the next process in your chain of nightly batch processes begins. Use the `maintenance_tools` command-line tool or the `MaintenanceToolsAPI` web service to check the status of PolicyCenter batch processes.

See also

- “Disabling the PolicyCenter Scheduler” on page 110
- “Maintenance Tools Command” on page 159
- “Running Batch Processes Using Web Services” on page 96 in the *Integration Guide*

Exclude Certain Batch Processes from Running During Your Nightly Batch Window

You may want to schedule some PolicyCenter batch processes to run periodically throughout the business day. For example, the default configuration of PolicyCenter schedules the `ActivityEsc` batch process to run every 30 minutes. Exclude running such batch processes periodically during your nightly batch processing window. Instead, wait until the end of the batch window to run them, For example, schedule the `ActivityEsc` to run every 30 minutes except during your nightly batch window. Alternatively, run such batch processes at prescribed places in your chain of nightly batch process.

Scheduling Batch Processes for Specific Environments

You can define a schedule for each batch process for different environments. To specify an environment for the process schedule, include the `env` attribute on the `ProcessSchedule` element.

```
<ProcessSchedule process="process_code" env="environment">
  <CronSchedule schedule_attributes/>
</ProcessSchedule>
```

As a consequence, you can now have different results for batch processing based on environment.

Disabling the PolicyCenter Scheduler

You can choose to use your own mechanism for running PolicyCenter system processes. For example, you can use the PolicyCenter API or command-line utilities to run the processes, and you can use your own scheduling application to trigger their execution. If you do this, you might choose to disable the internal PolicyCenter scheduler. To disable the internal scheduler, set the `SchedulerEnabled` configuration parameter to `false`:

```
<param name="SchedulerEnabled" value="false" />
```

Performing Custom Actions After Batch Processing Completion

You can use the Process Completion Monitor process to launch custom actions after a work queue or batch process completes a batch of items. For example, you might want to start the writer of a follow-on work queue during nightly batch processing.

The Process Completion Monitor process runs at schedulable intervals and examines the `ProcessHistory` for all completed work queues and batch processes. For completed work queues, the Process Completion Monitor also checks that all work items either completed or failed. If the process completed and no available or checked out work items remain, the Process Completion Monitor calls the `IBatchCompletedNotification` plugin to permit user code to react to the completion. The Process Completion Monitor sets `ProcessHistory.NOTIFICATIONSENT` to `true` for a completed process to ensure that it invokes the `IBatchCompletedNotification` plugin a single time only for any given process.

The `IBatchCompletedNotification` plugin has a `completed` method that you override to perform specific actions if a work queue or batch process completed a batch of work. The parameters of the `completed` method are the `ProcessHistory` and the number of failed items. A work queue batch is considered complete if no work items remain on the queue, other than work items that failed. A batch process batch is complete if the process stopped and its process history is available.

To schedule the Process Completion Monitor

1. In Studio expand **configuration** → **config** → **scheduler** and open `scheduler-config.xml`.
2. Add the following `ProcessSchedule` element:

```
<ProcessSchedule process="ProcessCompletionMonitor">
  <CronSchedule minutes="*/5"/>
</ProcessSchedule>
```

3. Save your changes.

This procedure schedules the Process Completion Monitor to run every five minutes. You can alter the schedule by modifying the `CronSchedule` element. See “Defining a Schedule Specification” on page 107.

To create a class to implement the IBatchCompletedNotification interface

1. In Studio, expand **configuration** → **gsrc**. If you have already created a package for your plugins within **gsrc**, navigate to that package.
2. Right-click **gsrc** or your custom package and click **New** → **Package**.
3. Enter a package name, such as `workqueue`.
4. Right-click the new package and click **New** → **Gosu Class**.
5. Enter a Name for the Gosu Class, such as `MyBatchNotificationPlugin`.
6. Click OK.
7. Define your Gosu class, using the following framework as an example:

```
package myCompany.plugin.workqueue
uses gw.plugin.workqueue.IBatchCompletedNotification

class MyBatchNotificationPlugin implements IBatchCompletedNotification {

  construct() {
  }

  override function completed(batch : ProcessHistory, numFailed : int) {

  //do something

  return;
  }
}
```

8. Right-click the GS file for your plugin and click Compile to ensure your plugin compiles successfully.
9. Save your changes.

To register the custom batch notification plugin

1. In Studio, expand **configuration** → **config** → **Plugins**.
2. Right-click **registry** and click **New** → **Plugin**.
3. In the **Plugin** dialog, enter the name of your plugin, such as `MyBatchNotificationPlugin`.
4. In the **Plugin** dialog, click the **...** button.
5. In the **Select Plugin Class** dialog, type `IBatchCompletedNotification` and select the **IBatchCompletedNotification** interface.
6. In the **Plugin** dialog, click OK. Studio creates a GWP file under **Plugins** → **registry** with the name you entered.
7. Click the **Add Plugin** icon (a plus sign) and select **Add Gosu Plugin**.
8. For **Gosu Class**, enter your class, including the package.
9. Save your changes.

See also

- “Process Completion Monitor” on page 119

Troubleshooting Work Queues and Batch Processes

This section discusses topics that are helpful in troubleshooting the management of batch processes.

Tuning Your Batch Process Schedule

To improve the performance of the batch operations, stagger the schedule so processes do not run at the same time. See “Scheduling Work Queues and Batch Processes” on page 107. You can also run these processes manually from the command-line.

Running Batch Processes from the Command Line

You can use the `maintenance_tools` command-line tool to run any of the batch processes. Run the following command:

```
maintenance_tools –password password –startprocess process_code
```

Monitoring Batch Processes

To check the status of a batch process, run the following command:

```
maintenance_tools -password password –processstatus process_code
```

For work queues, this command returns the status of the writer. It is possible that the writer has completed adding work items to the queue but workers have not finished processing them all.

You can also terminate a running process with the following command:

```
maintenance_tools -password password –terminateprocess process_code
```

Troubleshooting Work Queues

Workers can encounter problems in processing that cause the worker to fail before completing items that the worker has checked out. For example, a server might die, killing its workers in the middle of processing. This can result in orphan work items. Orphans are created if a worker has an item checked out but does not complete processing a item within the allotted `progressinterval` (current time - `LastUpdateTime` > `progressinterval`). Workers treat orphans just as they do `available` items. The next worker that encounters the orphan item in the table, adopts it for processing and resets the `LastUpdateTime`, `CheckedOutBy`, and `Status` fields on the orphan work item.

If a work queue is experiencing a large number of orphans, review log files to locate timeouts during processing. For example, a timeout might be caused by a worker waiting for an external server to return a value. If the log contains these type of timeouts, adjust the `progressinterval` value to give workers more processing time.

Sometimes, a problem inherent in the item itself causes processing of the item to fail. For example, an exception is thrown. In such cases, the worker stops processing the item and goes on to the next. The item becomes orphaned and the next worker attempts to process it. In this way, a work queue attempts to process each item multiple times up to a limit set by the item’s configuration. If the work queue exceeds the item’s limit, PolicyCenter changes the item’s status to `failed`. Workers ignore `failed` items and no longer attempt to process them.

To remove failed work items, run the Purge Failed Work Items process. Run the Purge Failed Work Items process twice to complete the purge. The first run sets the date on the failed work items and the second run performs the actual purge. See “Purge Failed Work Items” on page 120.

List of Work Queues and Batch Processes

PolicyCenter has a number of work queues and batch processes for different kinds of batch processing supported by PolicyCenter. The following topics describe the work queues and batch processes provided with PolicyCenter.

Running Batch Processing on a Schedule

You can schedule many of the work queues and batch processes provided with PolicyCenter to run at regular intervals. To configure the schedule for running a work queue or a batch process, edit the schedule configuration file. In the **Project** window in Studio, navigate to **configuration** → **config**, and then open `schedule-config.xml`.

Running Batch Processing from Server Tools

You can run many of the work queues and batch processes provided with PolicyCenter on demand from the **Work Queue Info** or **Batch Process Info** pages. These pages are available from the **Server Tools** tab. To access the tab, press `ALT+SHIFT+T`. Then, click **Work Queue Info** to access the **Work Queue Info** page, or click **Batch Process Info** to access the **Batch Process Info** page.

Users with the `internaltools` permission can access the **Server Tools** tab. The Admin role has `internaltools` permission by default.

Running Batch Processing by using Maintenance Tools

You can run many of the work queues and batch processes provided with PolicyCenter on demand from the command line by using the `maintenance_tools` command. If a work queue or batch process is not available from the **Server Tools** tab, you can run it only by using the `maintenance_tools` command.

See also

- “Scheduling Work Queues and Batch Processes” on page 107
- “Work Queue Info” on page 135
- “Batch Process Info” on page 134
- “Running Batch Processes” on page 103

Activity Escalation

Code – `ActivityEsc`

Activities in PolicyCenter can have an escalation date. The escalation date is the date on which PolicyCenter marks an open or overdue activity as requiring urgent attention. Activity Escalation batch processing operates on activities that need to be escalated due to the following criteria:

- The activity has an escalation date.
- The escalation date has passed.
- The activity has not already been escalated.

For qualifying activities, the Activity Escalation process marks the activity as escalated and calls the activity escalation rules to determine any actions.

Use the **Administration** tab to define escalation values for an activity pattern. Select an activity from the **Activity Patterns** page and click **Edit**.

Typically, you set deadlines by using **Escalation Days.** However, you can set them in **Escalation Hours** or both depending on business practices.

If you set your deadlines only in days, then run Activity Escalation batch processing no more frequently than daily. If your deadlines are shorter, run process more frequently to take action on overdue activities in a timely manner.

Do not delete an activity pattern that is in production as there could be old activities tied to the pattern. Instead, change the activity pattern setting **Automated only** to **Yes** to prevent users from accessing it from the user interface.

By default, PolicyCenter runs Activity Escalation batch processing every 30 minutes. Change this schedule as needed.

See also

- "Activities" on page 365 in the *Application Guide*
- "Defining Activity Patterns" on page 405 in the *Configuration Guide*
- "Activity Escalation Rules" on page 38 in the *Rules Guide*

Apply Pending Account Data Updates

Code – `ApplyPendingAccountDataUpdates`

Apply any of the pending updates to account data.

See also

- "Revisioning Contact Information in Policies" on page 376 in the *Application Guide*

Archive Policy Terms

Code – `ArchivePolicyTerm`

Archive Policy Terms batch processing archives policy terms. The process calls the `IPCArchivingPlugin` to determine whether a policy can be archived.

For a policy period to be eligible for archiving, the server time must have reached the `PolicyTerm.NextArchiveCheckDate` date:

- For more information on policy term eligibility, see "Selecting Policy Terms for Archive Eligibility" on page 448 in the *Configuration Guide*.
- For information on the archive work queue, and those policy periods that the archive process archived, excluded, or skipped, see "Archive Info" on page 144.

After running Archive Policy Terms batch processing, Guidewire recommends that you update database statistics. This process makes large changes to the tables. Updating database statistics enables the optimizer to pick better queries based on more current data. For instructions to gather database statistics, see "Configuring Database Statistics" on page 42.

See also

- "More Information on Archiving" on page 325 in the *Application Guide*

IMPORTANT Guidewire strongly recommends that you contact Customer Support before implementing archiving.

Audit Task

Code – `AuditTask`

Audit task monitor.

Bound Policy Exception

Code – `BoundPolicyException`

Runs policy exception rules on every bound `PolicyPeriod` that has not had exception rules run on it for more than the number of days defined by `BoundPolicyThresholdDays` in `config.xml`. The `PolicyPeriod` must be either in force or expired within the last year.

See also

- see "Policy Exception Rules" on page 39 in the *Rules Guide*

Clear Policy Renewal Check Dates

Code – `PolicyRenewalClearCheckDate`

This batch process clears (null out) the existing check date for all policies through a single direct database update statement. Because this is a direct update statement, this batch process is only available in maintenance mode (a run level of `NO_DAEMONS` or lower). Only run this process when:

- The configuration of automated policy renewals has changed significantly enough that there is a risk that some policies will be picked up unacceptably late for renewal. Ultimately, the `PolicyRenewalPlugin` controls the lead time any individual policy needs for renewal. By default, it depends on the code of that plugin and the `NotificationConfig` system table as accessed through the `NotificationConfigPlugin`. As you can overwrite the functionality of either plugin, which types of changes might significantly change the renewal start date will vary.
- When bringing the server up after the configuration update has been applied. Bring the server up to a maintenance mode (`NO_DAEMONS` or lower), run this batch process once, then bring the server all the way up.

The danger of running this process at other times is that it issues a direct SQL update, and thus does not update bean version numbers or any other domain logic. If this process were executed while batch processes and users are running on the system, the actual renewal check dates could be polluted by some new and some old values.

This process is not available from the user interface and cannot be scheduled.

Closed Policy Exception

Code – `ClosedPolicyException`

Runs policy exception rules on every closed `PolicyPeriod` that has not had exception rules run on it for more than the number of days defined by `ClosedPolicyThresholdDays` in `config.xml`.

See also

- "Policy Exception Rules" on page 39 in the *Rules Guide*

Data Distribution

Code – `DataDistribution`

Calculates the distribution of data in the PolicyCenter database. This process cannot be scheduled. You can run this process from PolicyCenter by pressing `ALT` + `SHIFT` + `T` to access the **Server Tools**. Then click **Info Pages** and select the **Data Distribution** page. On that page, modify parameters for the data distribution batch job and click **Submit Data Distribution Batch Job**. See "Data Distribution" on page 149.

You can also run this process from the command line by using the `maintenance_tools` command or from a web service.

Database Consistency Check

Code – `DBConsistencyCheck`

Runs consistency checks on the PolicyCenter database.

Do not launch this process using the `maintenance_tools` command. You cannot specify which checks to run or which tables to run the checks against with the `maintenance_tools` command. Instead, use the **Consistency Checks** page to run the Database Consistency Check batch process from PolicyCenter. If you want to schedule this batch process, use the following command:

```
system_tools -user user -password password -checkdbconsistency
```

You can specify tables and checks using this command with two optional arguments:

```
-checkdbconsistency tableSelection checkTypeSelection
```

The *`tableSelection`* argument can be specified as:

- `all` – Run consistency checks on all tables.
- *`table name`* – The name of a single table on which to run checks.
- `tg.`*`table group name`* – The name of a table group. Table groups are defined in `config.xml`. For more information, see "Defining Table Groups" on page 69 in the *Installation Guide*.
- `@`*`file name`* – A file name with one or more valid table names or table group names entered in comma-separated values (CSV) format. Prefix table group names with `tg.`, such as `tg.MyTableGroup`. You can combine table groups and individual table names in the same file.

The *`checkTypeSelection`* can be specified as:

- `all` – Run all consistency checks on the specified tables.
- *`check name`* – The typecode value of a single consistency check to run.
- `@`*`file name`* – A file name with one or more valid consistency check names entered in comma-separated values (CSV) format.

If you specify one optional argument, you must specify the other.

The Database Consistency Check process runs only on the batch server.

See also

- "Checking Database Consistency" on page 40 for an overview of database consistency checks
- "Consistency Checks" on page 147 for details of the Consistency Checks page in PolicyCenter
- "System Tools Command" on page 162 for an explanation of command line options

Database Statistics

See "Configuring Database Statistics" on page 42.

Deferred Upgrade Tasks

Code – `DeferredUpgradeTasks`

Deferred Upgrade Tasks batch processing is run following an upgrade of a database for a PolicyCenter environment with the archiving feature enabled. See "Deferring Creation of Nonessential Indexes" on page 168 in the *Upgrade Guide*.

Extract Rating Worksheets

Code – `ExtractWorksheets`

Extract Rating Worksheets batch processing extracts rating worksheet data from worksheet container (`WorksheetContainer`) objects to files in a specified directory on the batch server. The process also marks `WorksheetContainer` objects for purging.

In the base configuration, Extract Rating Worksheets is disabled. After you enable it, you can run it manually or schedule it.

See also

- "Extract Rating Worksheets Batch Process" on page 544 in the *Configuration Guide*
- "Purge Rating Worksheets" on page 121

Form Text Data Delete

Code – `FormTextDataDelete`

Deletes orphaned, purged, or archived `FormTextData` entities. The `FormTextData` object stores the text data that makes up the XML data for a `Form`.

Geocode Writer

Code – `Geocode`

Searches for new addresses to geocode. This batch process runs periodically to update geocoding information on user contact primary addresses and account locations. The `UserContact` entity represents a PolicyCenter user.

By default, this schedulable process is disabled.

Group Exception

Code – `GroupException`

Runs the group exception rule sets on all groups in the system. By default, this process runs daily at 4:00 a.m.

Impact Testing Export

Code – `ImpactTestingExport`

Impact Testing Export batch processing exports test periods to an Excel file whenever you click **Create Excel Export File** on the **Impact Results** screen.

Impact Testing Test Case Preparation

Code – `ImpactTestingTestPrep`

This process generates baseline policy periods on the selected policies whenever you click **Create Baselines** from the **Create Baseline** screen.

Impact Testing Test Case Run

Code – `ImpactTestingTestRun`

This process generates test policy periods rated using the selected rate books whenever you click **Quote Test Periods** from the **Testing Periods** screen.

Job Expire

Code – `JobExpire`

Expires a job if no action has been taken upon it for a configured period of time. By default, this process runs daily at 6:00 a.m.

Job Expire batch processing changes jobs from `New`, `Draft`, or `Quote` status to `Expired`. In the default configuration, the process expires submissions in these statuses that are at least seven days past the effective date of the policy. You can configure the expiration threshold as the number of days past the effective date or the creation date.

You can enable expiration for other job types.

To configure the expiration effective date threshold

1. In Studio, expand **configuration** → **config** and open `config.xml`.
2. Search for `JobExpirationEffDateThreshold`.
3. Change the value to the number of days past the effective date after which a job can be expired.

To configure the expiration create date threshold:

1. In Studio, expand **configuration** → **config** and open `config.xml`.
2. Search for `JobExpirationCreateDateThreshold`.
3. Change the value to the number of days past the creation date after which a job can be expired.

   Note: Setting `JobExpirationCreateDateThreshold` to a negative number effectively disables create date checking, since no job can be created with a future create date. However, the `JobExpirationEffDateThreshold` check is still active.

Job Expire batch processing examines all jobs meeting the date criteria. but only expires those jobs for which `job.canExpireJob()` returns `true`.

To enable expiration for a job type

1. In Studio, expand **configuration** → **config** and open `config.xml`.
2. Search for `JobExpireCheck<`*`JobType`*`>`, for example, `JobExpireCheckAudit`.
3. Change the value from `false` to `true`.
4. (Audit only) Open `AuditProcess.gs` in the `gw.job` package. Modify the `canExpireJob` method to return `true` instead of `false`.

   Performance of queries for jobs will be improved if all applicable jobs can be expired. Business requirements often do not permit expiration of Audit jobs. As a result `canExpireJob()` will need to be overridden for Audit jobs if they are to be expired.

See also

- “Configuring Policy Transactions” on page 78 in the *Application Guide*
- “Job Expiration Parameters” on page 62 in the *Configuration Guide*

Open Policy Exception

Code – `OpenPolicyException`

Runs policy exception rules on every open, unlocked `PolicyPeriod` that has not had exception rules run on it for more than the number of days defined by `OpenPolicyExceptionThresholdDays` in `config.xml`.

See also

- “Policy Exception Rules” on page 39 in the *Rules Guide*

Overdue Premium Report

Code – `OverDuePremiumReport`

Overdue premium report monitor.

Phone Number Normalizer

Code – `PhoneNumberNormalizer`

Runs the registered plugin that implements the `IPhoneNumberNormalizer` interface. See “Upgrading Phone Numbers” on page 192 in the *Upgrade Guide*.

Policy Hold Job Evaluation

Code – `PolicyHoldJobEval`

Evaluates each job against the policy holds blocking it.

Policy Renewal Start

Code – `PolicyRenewalStart`

Policy renewal start monitor. By default, this process runs daily at 1:00 a.m.

Populate Search Columns

Code – `PopulateSearchColumns`

Populates denormalized `searchColumn` columns from their designated `sourceColumn` columns.

This process can be run only from the `maintenance_tools` command or using a web service.

See also

- “The <searchColumn> Subelement” on page 167 in the *Configuration Guide*

Premium Ceding

Code – `PremiumCeding`

Reinsurance ceding of premium.

See also

- “Reinsurance Ceding Plugin” on page 414 in the *Integration Guide*

Process Completion Monitor

Code – `ProcessCompletionMonitor`

You can use the schedulable Process Completion Monitor process to launch custom actions after a batch process completes. See “Performing Custom Actions After Batch Processing Completion” on page 110.

Process History Purge

Code – `ProcessHistoryPurge`

Purges batch process history data from the process history table. A large number of history records in the database can slow performance when a user accesses the **Batch Process Info** or **Work Queue Info** system tools.

Before you run Process History Purge, remove `WorkItem` and `InstrumentedWorkerTask` entities that reference `ProcessHistory`. Run the Purge Failed Work Items and Work Queue Instrumentation Purge processes to remove these entities. Run the Purge Failed Work Items process twice to complete the purge. The first run sets the date on the failed work items and the second run performs the actual purge.

The BatchProcessHistoryPurgeDaysOld parameter in config.xml specifies the number of days to retain batch process history. Set BatchProcessHistoryPurgeDaysOld to be 5 days greater than InstrumentedWorkerInfoPurgeDaysOld and not less than 35 days. For example, if InstrumentedWorkerInfoPurgeDaysOld equals 40, then set BatchProcessHistoryPurgeDaysOld to 45. This guarantees that referring work queue history records and failed work items are deleted by the Purge Failed Work Items and Work Queue Instrumentation Purge processes. Work queue history records and failed work items that refer to a batch process history record must be deleted before Process History Purge can delete the record.

Purge

Code – Purge

Purges jobs and prunes policy periods that meet the purge and prune criteria. This process deletes jobs and other entities from the database.

See also

- "Purge Batch Process" on page 462 in the *Configuration Guide*

Purge Cluster Members

Code – PurgeClusterMembers

Purges ClusterMemberData entities that have a LastUpdate value prior to the current date minus the value of the ClusterMemberPurgeDaysOld parameter.

This process can be scheduled or run from the user interface.

Purge Failed Work Items

Code – PurgeFailedWorkItems

Purges failed work items from all work queues. Run the Purge Failed Work Items process twice to complete the purge. The first run sets the date on the failed work items and the second run performs the actual purge.

Purge Message History

Code – PurgeMessageHistory

Purges old message records from the message history table. The KeepCompletedMessagesForDays parameter in config.xml specifies how many days a message can remain in the message history table before this process removes the message.

Purge Old Transaction IDs

Code – PurgeTransactionIDs

Purges TransactionId entities that have a CreationDate prior to the current date minus the value of the TransactionIdPurgeDaysOld parameter. The TransactionIdPurgeDaysOld parameter can be configured in the config.xml file.

Purge Orphaned Policy Periods

Code – PurgeOrphanedPolicyPeriod

Purges policy periods orphaned as a result of preemption. This batch process deletes policy periods and other entities from the database.

See also

- "Purge Orphaned Policy Periods Batch Process" on page 466 in the *Configuration Guide*

Purge Profiler Data

Code – `PurgeProfilerData`

Purges data gathered by the profiler. The `ProfilerDataPurgeDaysOld` parameter in `config.xml` specifies how many days to retain profiler data before this process removes the data.

Purge Rating Worksheets

Code – `purgeworksheets`

This process removes worksheet container (`WorksheetContainer`) objects that are marked for purging and are on policies whose jobs have been closed more than a specified number of days. In the base configuration this period is 90 days. In the base configuration, Extract Rating Worksheets batch processing marks the worksheets container objects for purging.

In the base configuration, this batch process is not scheduled to run on a regular basis. You can run it manually or schedule it.

See also

- "Purge Rating Worksheet Batch Process" on page 544 in the *Configuration Guide*
- "Extract Rating Worksheets" on page 116

Purge Workflow

Code – `PurgeWorkflows`

Purges completed workflows after resetting any referenced workflows. The `WorkflowPurgeDaysOld` parameter in `config.xml` specifies how many days to retain workflow data before this process removes the data.

This process executes `gw.processes.PurgeWorkflow.gs`. You can modify this Gosu class to customize the functionality of this batch process.

Purge Workflow Logs

Code – `PurgeWorkflowLogs`

Purges logs of completed workflows. The `WorkflowLogPurgeDaysOld` parameter in `config.xml` specifies how many days to retain workflow logs before this process removes the logs.

This process executes `gw.processes.PurgeWorkflowLogs.gs`. You can modify this Gosu class to customize the functionality of this batch process.

Reset Purge Status and Check Dates

Code – `ResetPurgeStatusAndCheckDates`

Resets the purge status and purge or prune dates on the jobs. Run this process if and when you have a need to reset the purge status and purge date.

For each job, this batch process:

- Sets the `Job.PurgeStatus` property to `Unknown`.
- Sets the `Job.NextPurgeCheckDate` to `null`.

For example, if a job has a `PurgeStatus` of `NoActionRequired` or `Pruned`, the batch process resets the `PurgeStatus` to `Unknown`.

Retire Activities

Code – `ActivityRetire`

Retires `Activity` records that are either canceled or dismissed.

Retrieve Policy Terms

Code – `RestorePolicyTerm`

Run this batch process to retrieve archived policy terms that have been marked for retrieving. A user can request that an archived policy term be retrieved. Retrieving a policy term generates an activity for the user who requested the retrieve.

See also

- "Retrieving Archived Policies" on page 324 in the *Application Guide*

Solr Data Import

Code – `SolrDataImport`

Run this batch process to test the operation of the free-text batch load command, especially its embedded SQL query. This batch process is intended only for development mode. Do not run this process in production to load and reindex the Guidewire Solr Extension. Instead, run the free-text batch load command on the host where the Guidewire Solr Extension resides.

See also

- "Free-text Batch Load Command" on page 173
- "Free-text Search System Architecture" on page 342 in the *Configuration Guide*

Team Screens

Code – `TeamScreens`

This batch process collects statistics for the **Team** tab screens.

See also

- "Team Management" on page 673 in the *Application Guide*
- "Configuring the Team Tab" on page 515 in the *Configuration Guide*

User Exception

Code – `UserException`

Runs the user exception rule sets on all users in the system. By default, this process runs daily at 3:00 a.m.

See also

- "User Exception Rules" on page 41 in the *Rules Guide*

Workflow

Code – `Workflow`

This work queue wakes and runs workflow worker threads.

See also

- "Guidewire Workflow" on page 373 in the *Configuration Guide*

Work Item Set Purge

Code – `WorkItemSetPurge`

Purges work item sets from the database. The `BatchProcessHistoryPurgeDaysOld` parameter in `config.xml` specifies the number of days to retain work item sets. This parameter also configures how many days to retain process history records, which are removed by the separate Process History Purge process.

Work Queue Instrumentation Purge

Code – `WorkQueueInstrumentationPurge`

Purges instrumentation data for work queues. The `InstrumentedWorkerInfoPurgeDaysOld` parameter defines how long to retain work queue instrumentation data. You can download work queue instrumentation data for a work queue by clicking **Download History** for the work queue on the **Work Queue Info** page.

See also

- "Work Queue Info" on page 135

Other Processes

Some processes included with PolicyCenter are used by PolicyCenter internally or are not used at all. You cannot run these processes from PolicyCenter, `maintenance_tools`, or an API.

Unused Processes

Some Guidewire platform processes that are not used by PolicyCenter are included in `PolicyCenter/modules/pl/config/metadata/BatchProcessType.tti`. You can ignore these processes. The processes include:

- Bulk Purge
- Contact Auto Sync
- Staging Table Delete Excluded Rows
- Staging Table Encryption
- Staging Table Integrity Check
- Staging Table Load
- Staging Table Populate Exclusion Table
- Staging Table Statistics
- Stat Report Writer

Internal Processes

Some Guidewire platform processes are used by PolicyCenter internally. These processes are run by PolicyCenter only to generate database performance reports. You cannot run these processes separately.

- Microsoft DMV Report
- Oracle AWR Report

chapter 9

chapter 10

Configuring Guidewire Document Assistant

PolicyCenter provides the Document Assistant Java applet to enable client-side creating, viewing, editing and uploading of files. Document Assistant can be disabled in the server configuration, in which case all such operations are handled directly by the browser.

Document Assistant supports automatic creation of new documents from custom document templates. Document Assistant opens and displays documents through Windows rather than returning the documents directly to the browser. Document Assistant directly opens documents that have an allowed file type. See “Document Assistant Supported File Types” on page 129.

For more information about document management, see “Document Management” on page 191 in the *Integration Guide*.

This topic includes:

Enabling Guidewire Document Assistant

Guidewire Document Assistant is disabled by default. The following procedure enables the Guidewire Document Assistant.

To enable Document Assistant

1. Enable Document Assistant from the `config.xml` file by setting:
   ```
   <param name="AllowDocumentAssistant" value="true"/>
   ```
2. Configure PolicyCenter to display document **Edit** and **Upload** buttons by setting:
   ```
   <param name="DisplayDocumentEditUploadButtons" value="true"/>
   ```
3. Save `config.xml`.

Support for Document Management Systems

PolicyCenter provides a **Documents** section for certain entity types such as a Policy. The **Documents** section lists a set of document references that PolicyCenter assumes your company stores externally, perhaps in a document management system. Guidewire provides a reference implementation for document management. This implementation assumes that documents reside on the file system. You can view this reference implementation in `PolicyCenter/java-api/examples/src/examples/plugins/document`. If you do not see the `java-api` directory, run the following command from the PolicyCenter `bin` directory:

```
gwpc regen-java-api
```

You can create a custom integration with a document management system. For details, see “Document Management” on page 191 in the *Integration Guide*.

Client Configuration Requirements

The Document Assistant is a signed Java Web Start (JWS) applet requiring Java Runtime Environment 7 (JRE 7) or later on the client machine. The applet is deployed in the browser by the Java Next-Generation Plugin using the Java Network Launch Protocol (JNLP).

The Document Assistant applet is inserted into the browser Document Object Model (DOM) and loaded on demand for the first operation requiring the Document Assistant. The applet and its resources are signed with the Guidewire code signing certificate, and the user is required to accept this certificate at least once. If the user elects to install the Guidewire certificate permanently in their JRE key store, they will never be prompted for authorization again. The resources are signed with a timestamp signature from a universally-recognized certificate authority. Therefore, the applet and resources will still be considered valid even after the certificate originally used to sign them has expired.

The user web client must have Java installed. For the list of supported browsers, Java versions, and operating systems, see the *Guidewire Platform Support Matrix*. The *Guidewire Platform Support Matrix* is available from the Guidewire Resource Portal at `https://guidewire.custhelp.com/app/resources/products/platform`.

The Java Console can be used to examine diagnostic messages from the applet. Tracing and logging can be enabled through the console or with the control panel. The Java Control Panel can be used to display the currently installed version of the Document Assistant. It may also be used to examine the details of the Guidewire code signing certificate. Other messages can be seen in the browser's error console.

Document Assistant supports client-side JScript required for some templates, such as the MailMerge template. The MailMerge template is a Guidewire-supplied reference implementation. To support client-side JScript, Document Assistant creates a temporary file containing the required JScript and then runs it using Windows Script Host. To support this mechanism, the `.js` extension on the user’s environment must map to Windows Script Host (`Windows\System32\wscript.exe`).

The Document Assistant is a JWS applet that uses LiveConnect (a JavaScript to Java bridge) calls to interact with the main application in the browser. The JRE security settings must allow both the applet itself to run and the LiveConnect calls to and from the applet. Document Assistant is code signed by Guidewire to allow these permissions. Document Assistant typically requires no security changes if run against JRE 7u51 or higher and the current security baseline or higher.

Guidewire recommends that end users stay current with Java security updates for their client JRE. Otherwise it is possible for the browser to block Document Assistant from running. Techniques described in this topic make it possible to run Document Assistant in scenarios in which it might otherwise be blocked. Deployment rule sets are intended for site administrators. Exception site lists and security sliders on the Java Control Panel are intended for end users, though it is possible for administrators to control those as well. If using exception site lists or security sliders, the browser will still report security warnings if the client java version is below the security baseline, such as:

- Java version is out of date
- Potentially unsafe components
- Warnings about the application running with unrestricted access
- Warnings about the web site referencing JavaScript code that is requesting access and control of a Java application on the web page

Creating a Deployment Rule Set

If end users are using Java 7 update 40 or above, the site administrator can create a deployment rule set to give users permission to run Guidewire Document Assistant. Without a deployment rule set, a desktop user can receive security warnings or the Guidewire Document Assistant could be blocked, depending on the Java Control Panel security level. Deployment rule sets were introduced in Java version 7 update 40.

WARNING Using a deployment rule set opens a potential security hole. The deployment rule allows any applet from the PolicyCenter URL to run, signed or not, rather than just the signed Document Assistant applet. Deployment rule sets bypass all security checks and are chained from first to last until there is a match. So take extreme caution in ensuring that you do not give permissions to more applets than intended. You cannot use a certificate hash-based rule for the applet because a certificate hash-based rule only grants permission to the applet itself. It does not grant permission to the LiveConnect calls required by the applet.

To create a deployment rule set

1. Create a file called `ruleset.xml` with the following content:

```
<ruleset version="1.0+">

  <rule>
    <id location="URL for PolicyCenter"/>
    <action permission="run" />
  </rule>

  <!-- Because this is both blank and shown last, it will be the default policy. -->
  <rule>
    <id />
    <action permission="default"/>
  </rule>

</ruleset>
```

2. Package `ruleset.xml` into a JAR called `DeploymentRuleSet.jar`.

```
jar -cvf DeploymentRuleSet.jar ruleset.xml
```

3. Sign `DeploymentRuleSet.jar` with a valid code signing certificate.

```
jarsigner -keystore jks-keystore -storepass keystore-password -tsa URL for Time Stamping Authority
DeploymentRuleSet.jar keystore-alias
```

4. Copy `DeploymentRuleSet.jar` to *`Windows directory`*`\Sun\Java\Deployment`, for example, `C:\Windows\Sun\Java\Deployment`.

For more information about deployment rule sets, refer to `https://blogs.oracle.com/java-platform-group/entry/introducing_deployment_rule_sets`.

Creating an Exception Site List

If end users are using Java 7 update 51 or above, you can create an exception site list on end user machines to allow users to run Guidewire Document Assistant. Refer to the following URL for instructions:

`http://java.com/en/download/faq/exception_sitelist.xml`

Enter the PolicyCenter URL for the **Location** field when setting up the exception site list.

Setting Security Levels in the Java Control Panel

You can set the security level in the end user Java Control Panel to Medium to avoid security warnings. Refer to the following URL for instructions:

`http://www.java.com/en/download/help/jcp_security.xml`

Guidewire Document Assistant Configuration Parameters

You can set the following configuration parameters related to document management:

Parameter	Description
`AllowDocumentAssistant`	Whether to allow document management controls in the PolicyCenter interface. Setting this to `false` removes all controls from the interface, which results in reduced functionality. If `false`, this turns the Guidewire Document Assistant control off entirely and also forces the following parameters to be `false`: • `DisplayDocumentEditUploadButtons` • `UseDocumentAssistantToDisplayDocuments` When Document Assistant is disabled, users can still view documents by downloading the document in the browser. Default: `false`
`DisplayDocumentEditUploadButtons`	Whether the Documents list displays **Edit** and **Upload** buttons. Set this to false if the `IDocumentContentSource` integration mechanism does not support it. Default: `true`
`DocumentAssistantJNLP`	The relative or absolute URL for the Document Assistant JNLP launch file. If specified as a relative URL, the URL is relative to the PolicyCenter web server host. Default: `/jnlp/gw/documentassistant/DocumentAssistant.jnlp`
`DocumentContentDispositionMode`	The `Content-Disposition` header setting to use any time that PolicyCenter returns document content directly to the browser. The `Content-Disposition` header setting specifies how the browser displays a document. This value can be either `inline` (the default) or `attachment`.
`DocumentTemplateDescriptorXSDLocation`	Name of the XSD file PolicyCenter uses to validate document template descriptor XML files. Specify this location relative to the following directory: `modules/configuration/config/resources/doctemplates` PolicyCenter loads the XSD file from `PolicyCenter/modules/configuration/config/resources/doctemplates`.

`MaximumFileUploadSize`	Specifies the maximum file size in megabytes that a user can upload to the server. Any attempt to upload a file larger than this fails. Since the server must handle the uploaded document, this parameter protects the server from possible memory consumption problems. Note: This parameter setting affects any imports managed through the PolicyCenter Administration tab. This specifically includes the import of administrative data and roles. Default: `20`
`UseDocumentAssistantToDisplayDocuments`	Whether to use the Guidewire Document Assistant control to display document contents. If `false`, PolicyCenter does not use the control and document contents return directly to the browser. Default: `true`

Document Assistant Supported File Types

The Document Assistant supports auto-launching of the following file types:

• AVI	• GIF	• MDI	• PNG	• RTF	• WAV
• BMP	• HTM	• MOV	• PPS	• RTX	• WMA
• CSV	• HTML	• MPEG	• PPT	• TIF	• XLS
• DOC	• JPG	• MPG	• PPTX	• TIFF	• XLSX
• DOCX	• LOG	• PDF	• PS	• TXT	• XML

For files with extensions not listed, Document Assistant uploads or downloads the file but does not launch the file. Use anti-virus and file system protection software to minimize the risk from downloaded files.

AVI and WMA files can contain security vulnerabilities. Guidewire strongly recommends that you update all client machines with video-playing software that contains the latest security patches. For specific information, see the following Microsoft web pages:

For AVI: `http://www.microsoft.com/technet/security/Bulletin/MS09-038.mspx`

For WMA: `http://www.microsoft.com/technet/security/bulletin/ms09-051.mspx`

You can customize the file types that Document Assistant handles by customizing the `WhitelistExtensions` file within the resources JAR. See “Customizing Document Assistant” on page 129.

Customizing Document Assistant

You can customize Document Assistant by creating a custom version of the resources JAR.

To customize Document Assistant

1. Rename `pl-documentassistant-resources__V<`*version*`>.jar` in `modules/configuration/deploy/jnlp/gw/documentassistant` to some other path under `modules/configuration/deploy/jnlp`. The new name must be of the form `<`*base name*`>__V<`*custom version*`>.jar` and be within `modules/configuration/deploy/jnlp`. Place the custom JAR in any subdirectory of `modules/configuration/deploy/jnlp`, including `gw/documentassistant`, so PolicyCenter can locate the resource. `<`*custom version*`>` must be different from `<`*version*`>`. For example, you could rename `pl-documentassistant-resources__V8.2.0.jar` to `pl-documentassistant-customized-resources__V8.2.0-C1.jar`.
2. Update `DocumentAssistantResources.jnlp` for the new custom version.

```
<information>
  <title>Document Assistant Customized Resources <custom version></title>
  <vendor>Guidewire Software, Inc. and <Company Name></vendor>
</information>
...
<resources>
...
  <jar href="pl-documentassistant-customized-resources.jar" version="<custom version>" main="false"/>
...
</resources>
```

3. If the new resource JAR is in a location different from the `DocumentAssistantResources.jnlp` file, that change must be included as well. For example, if you place the JAR in `deploy/jnlp/customized`:

```
<resources>
...
  <jar href="/pc/jnlp/customized/pl-documentassistant-customized-resources.jar"
   version="<custom version>" main="false" />
</resources>
```

4. Make changes to the new resources JAR contents, described in "Document Assistant Resource Jar Contents" on page 130. Then re-sign the JAR with a valid code signing certificate specific to your organization using `jarsigner` or equivalent JDK tools. It is critical that the signature replace any existing signature by Guidewire and be recognized by the end user browsers, or Document Assistant will not load properly.

5. Regenerate the PolicyCenter WAR or EAR file to copy your changes to the `webapp` area. During testing, the Java Control Panel can be used to clear the cache of old copies of changes rather than to constantly iterate on the custom version. Or you can disable caching during testing.

Document Assistant Resource Jar Contents

The `documentassistant` directory includes the following contents:

File	Description
`DocumentAssistant.jnlp`	Master JNLP launch file
`DocumentAssistantLogo.jpg`	Logo image in client Java Control Panel
`DocumentAssistantResources.jnlp`	Resources Extension JNLP launch file
`pl-documentassistant-applet__V<version>.jar`	JAR containing the Document Assistant applet and internal resources (read-only)
`pl-documentassistant-applet__V<version>.jar.pack.gz`	Pack200 compressed version of Document Assistant applet JAR (read-only)
`pl-documentassistant-resources__V<version>.jar`	JAR containing Document Assistant whitelist extensions and client-side document production scripts. You can customize these resources.

You can create a custom version of `pl-documentassistant-resources__V<version>.jar` with customized resources. The JAR file includes a `documentassistant` directory that contains an `unknown` and a `windows` directory. The `windows` directory includes the following resources:

File	Description
`ExcelMerge.js`	The `ExcelMerge.js` file contains the JScript that Document Assistant uses when handling an Excel document. You can customize the JScript by modifying this file.
`Utility.js`	The `Utility.js` file contains utility scripts used by Document Assistant.

File	Description
`WhitelistExtensions`	A plain-text file containing a dot-delimited list of file extensions that Document Assistant will open. You can add or remove extensions to this file to enable or disable that file type for Document Assistant. Edit the WhitelistExtensions file within the `documentassistant/windows` directory. The single-line compact formatting of the default file is not strictly required. Only the dot delimiter is required. Do not include comments or other extraneous text. Any addition to `WhitelistExtensions` must be added to the MIME configuration on the server. See "Adding a custom MIME type for Document Production" on page 212 in the *Integration Guide*. Removals from WhitelistExtensions do not need to be removed from the server MIME configuration. The server will accept uploads of that MIME type, but the Document Assistant will not automatically open or launch such files.
`WordMerge.js`	The `WordMerge.js` file contains the JScript that Document Assistant uses when handling a Word document. You can customize the JScript by modifying this file.

Disabling Guidewire Document Assistant

PolicyCenter disables Guidewire Document Assistant by default. If you enabled Document Assistant, you can later choose to disable it.

The following procedures disable the Guidewire Document Assistant.

To disable the Guidewire Document Assistant

1. Disable Guidewire Document Assistant from the `config.xml` file by setting:
   ```
   <param name="AllowDocumentAssistant" value="false"/>
   ```
2. Save `config.xml`.

chapter 11

Using Server and Internal Tools

This topic discusses how to use the **Server Tools** and **Internal Tools** for administrative tasks.

This topic includes:

Using the Server Tools

Guidewire provides **Server Tools** to assist you with certain server and database administration tasks. This section contains the following topics:

Overview of Server Tools

Typically, users must have the `internaltools` permission to access the **Server Tools** pages. The Superuser role has this permission by default. Alternatively, if the `EnableInternalDebugTools` parameter is set to true in `config.xml`, and the server is running in development mode, all users have access to the **Server Tools** pages. For more information about server modes, see "Server Modes and Run Levels" on page 58.

In addition to the **Server Tools**, there are a number of unsupported **Internal Tools**. Guidewire provides the **Internal Tools** for use during development only and does not support **Internal Tools** for production environments. See "Using the Internal Tools" on page 154.

Press `ALT+SHIFT+T` to display the **Server Tools** tab. At the top of each page in the title bar is a link that returns you to the main PolicyCenter interface. Alternatively, you can **Logout** from this page to exit PolicyCenter.

Batch Process Info

Use the **Batch Process Info** page to run and view information about PolicyCenter batch processes, including writer threads for work queues. This information includes the **Batch Process** name, **Description**, **Status**, **Last Run** time, **Last Run Status**, **Next Scheduled Run** time, and scheduling information. The **Cron-S M H DOM M DOW** schedule column header stands for seconds, minutes, hours, days of month, month, and day of week.

Note: You can run writers for work queues from the **Work Queue info** page or the **Batch Process Info** page.

To run a batch process, click **Run** in the **Action** column for the batch process. The **Run** button is enabled for all batch process types that belong to the `BatchProcessTypeUsage` category `UIRunnable`. To stop a batch process, click **Stop** in the **Action** column for the batch process.

To download a history for a batch process, click **Download History** in the **Action** column for the batch process. You can then specify a range of dates to include in the history information. The historical data includes the date and time that the process started and completed, the number of operations, and the number of failed items along with the failure reason.

Use the drop-down on the **Batch Process Info** page to filter the batch process list. You can set the filter to show **Any** processes, or only **Schedulable** or **Runnable** processes.

You can use the **Batch Process Info** page to view the history of a batch process. Select the batch process. The bottom pane provides **Chart** and **History** tabs. The **Chart** tab shows the execution time in seconds and the number of operations performed by the batch process over time. The **History** tab includes a table of records of past runs of the selected batch process. This **History** table includes the following information:

Column	Description
Started	The time that a batch process started.
Completed	The time that a batch process completed.
Ops	For batch processes that are work queue writers, **Ops** is the number of work items processed by a work queue. This number includes work items that failed.
Failed	For batch processes that are work queue writers, **Failed** is a counter that is incremented each time an exception is encountered while processing a work items. The work queue might attempt to process a failed work item multiple times. Therefore, the **Failed** and **Ops** numbers will not necessarily match the total number of work items.
Failure Reason	For batch processes that are work queue writers, **Failure Reason** is the reason that a work item failed processing.

You can configure this page to restrict the batch processes that can be run from this dialog. To change the configuration, edit the `ServerTools.pcf` page. Access `ServerTools.pcf` from Guidewire Studio at **configuration** → **Page Configuration** → **pcf** → **tools** → **ServerTools**. The `BatchProcessType` typelist lists the possible batch processes you can display.

Note: You cannot start multiple runs of custom batch processes that are designed to be non-exclusive from the **Batch Process Info** page. Instead, you must use the maintenance tools command to start multiple runs of non-exclusive custom batch processes.

See also

- "Maintenance Tools Command" on page 159
- "Scheduling Work Queues and Batch Processes" on page 107
- "Exclusive" on page 583 in the *Integration Guide*

Work Queue Info

Use the **Work Queue Info** page to control and view information associated with work queues. From this page, you can track work queues as they process information. Each work queue has both a writer and one or more workers. You can run the writer to add a batch to the work queue, and you can monitor the progress of the workers processing the batch. To work with this dialog, you must first select a **Work Queue**.

You can click **Download** to download work queue information. This information includes a summary of the work queues and detailed information for specific work queues. You can specify the maximum number of writers, executors, batches for each worker, and the number of hours for which to generate item distribution data in the report. The detailed information includes data for each worker by thread and by host, such as:

- How long the worker has been active
- How many items the worker processed
- Throughput (items processed per minute of execution) per thread and per host
- Start and end times for the worker
- Last wake up time
- Items processed per thread (cumulative, average, and maximum)
- Uptime (cumulative, average, and maximum)
- Execution time (cumulative, average, and maximum)

The report includes a view called **Work Queue Runs**. This view shows work queue statistics organized by writer run, including how long it took the writer to produce work items and how long the workers processed those items. The view can be confusing if the writer is run repeatedly before finishing the work items produced by the previous run.

To download a report on work queue instrumentation in CSV format, click **Download Raw Report**. The report contains time-sliced raw data from the `ProcessHistory` and `InstrumentedWorkerTask` tables in CSV format for analysis with third-party tools such as Microsoft Excel.

By default, PolicyCenter shows statistics **By Writers** associated with the queue. Use the **By Workers** tab to view the workers associated with the queue.

The top-level columns have the following meanings:

Column	Description
Work Queue	Name of the work queue.
Available	Number of work items available for processing.
Checked Out	Number of work items checked out by workers.
Failed	Number of work items that failed during processing.

Column	Description
Executors Running	Number of workers processing the work queue.
Writer Status	Status of the writers.
Actions	Actions that you can perform on the work queue. These include: **Run Writer –** Launches the writer to write work items for the work queue. **Notify Executor –** Wake workers by notifying the executor that there are items to process. See “Worker Thread Management” on page 106. **Stop Executor –** Stops the executor currently managing the work queue. **Restart Executor –** Restarts the executor. **Download History –** Downloads the historical instrumentation data for the work queue. You can clear the instrumentation data for all work queues by running the Work Queue Instrumentation Purge process.

An **Item Statistics** region provides information on work items.

The item counts in the **By Writers** columns are the counts generated by the writer. Each row represents one wake period for the writer. These values have the following meanings:

Column	Description
Process ID	The ID for the writer process.
Item Creation Time	The time at which the writer woke and began writing work items. The first item in the table for a queue has a creation time that matches the queue’s current **Last Execution Time for the Writer** value.
Scheduled	Whether the writer is scheduled.
Number of Items	The total work items in the queue regardless of status.
Worker End Time	The timestamp when the last worker completed work items.
Execution Time	The number of minutes the process has been executing.
Available	The total number of available items in the queue.
Checked Out	The number of items checked out by workers for processing.
Succeeded	The number of items that completed successfully.
Failed	The number of items that failed.

The **By Executors** tab lists active executors. The columns have the following meanings:

Column	Description
Hostname	The server on which the executor is running.
Max. Number of Workers	The maximum number of workers available to the executor.
Processed Items	Number of items processed by the workers.
Exceptions	Number of exceptions encountered during processing.
Failed items	Number of work items that failed processing.
Active	Specifies whether the executor is currently active.
Started	The timestamp when the executor was started.
Up For	The duration that the executor has been running.

Under the **By Executors** tab is a **By tasks** tab. The **By tasks** columns have the following meanings:

Column	Description
ID	The unique identifier of the task.
Writer	The identifier of the writer process.
Success	Whether the processing of work items by the worker was a success.
Checked out items	The number of work items the worker checked out.
Processed items	The number of work items the worker processed.
Exceptions	The number of exceptions, if any, encountered during item processing.
Orphans Reclaimed	The number of orphaned work items the worker has adopted for processing.
Failed items	The number of failed work items.
Skipped items	The number of items that were skipped.
Started	When the task started.
Ended	When the task ended.
Active	Whether the task is still active.
Consecutive Errors	If processing resulted in exceptions, the number of consecutive work items found that resulted in exceptions during processing by the worker.

The **Work items** tab lists work items. The columns have the following meanings:

Column	Description
ID	The unique identifier of the work item.
Create time	When the work item was created.
Update time	When the work item was last updated.
Available at	When the work item is available to be processed. This value is null for failed work items.
Server	The server which processed the work item.
Writer	The writer that created the work item.
Attempts	How many attempts a worker has made to process the item.

Management Beans

PolicyCenter includes management beans that represent different resources. You can use this dialog to view all and edit some of the attributes associated with these resources. The resources available from the dialog include:

Resource	Description
`com.guidewire.pl.system.monitor`	Tracks the sessions and connections associated with the PolicyCenter application server. If a user is logged in more than once, the user name has the number of sessions appended to it in parentheses.
`com.guidewire.pl.system.cluster`	Provides information about the servers in a cluster. This bean is only available if you run the server in a clustered environment.
`com.guidewire.pl.system.configuration`	View all and edit some of the configuration values in the system. You must have the `soapadmin` permission to change values associated with management beans.
`com.guidewire.pl.system.cache`	View all and edit some cache attributes. You must have the `soapadmin` permission to change values associated with management beans.

Viewing and Changing Configuration Parameters

By using the `com.guidewire.pl.system.configuration` bean you can view all configuration parameters. You can edit some of these parameters from the **Management Beans** page. See "Application Configuration Parameters" on page 35 in the *Configuration Guide* for descriptions of configuration parameters.

Viewing and Changing Caching Configuration Values

For `com.guidewire.pl.system.cache`, you can set the `MaxCacheSpace` and `StaleTimeMinutes` values. The `MaxCacheSpace` value is the maximum amount of memory to use for the cache. The `StaleTimeMinutes` value is the number of minutes an object can exist in the cache without being refreshed. If an object is in the cache longer than `StaleTimeMinutes`, PolicyCenter refreshes the object entry. Use the "Cache Info" on page 142 page to monitor cache performance. Then, adjust cache values based on your analysis of that information. See "Application Server Caching" on page 65 for more information.

Guidewire Profiler

The Guidewire Profiler provides information about the runtime performance of code. The Guidewire Profiler collects information for specific sections of code that can be defined by Guidewire developers and configured, to some extent, for specific implementations.

The Guidewire Profiler does not collect memory usage statistics. You can use a third-party tool to gather memory usage and garbage collection information. See "Analyzing Server Memory Management" on page 70.

Guidewire Profiler Terms and Concepts

The following sample code introduces key concepts for understanding the Guidewire Profiler:

In `ProfilerTag.java`:

```
ProfilerTag MYTAG = new ProfilerTag("MyTag");
```

In `MyCode.java`:

```
import gw.api.profiler.Profiler;
ProfilerFrame frame = Profiler.push(ProfilerTag.MYTAG);
try {
  myMethod(str);
} finally {
    Profiler.pop(frame);
}
```

Profiler Tag

Profiler tags represent sections of code that can be profiled by the Guidewire Profiler. A profiler tag is an alias for a piece of code in the Guidewire application for which you want to gather performance information.

Profiler tags are represented in the code by instances of the `gw.api.profiler.ProfilerTag` class. The constructor for the `ProfilerTag` takes a `String` parameter defining the `ProfilerTag` name. In this example, the profiler tag has the name `MyTag` and corresponds to the code invoked by the method `myMethod`.

It is better to create a static final `ProfilerTag` and preserve it, rather than create one each time you need it. Creating the tag incurs some slight performance cost.

Profiler Frame

A profiler frame contains information corresponding to a specific invocation of profiled code, such as its start and finish times. When `push()` is called on the profiler stack, a profiler frame is created and pushed onto the stack. When `pop()` is called on the profiler stack, the profiler frame is removed from the stack, but its information is stored so as to be available for future examination. Profiler frames are represented in the code by instances of `gw.api.profiler.ProfilerFrame`.

Profiler Stack

A profiler stack stores profiling information for a specific thread. See “Entry Points” on page 139. A profiler stack implements the standard `push()` and `pop()` functionality of a stack. The `push` and `pop` correspond to the beginning and end, respectively, of a piece of code represented by a profiler tag. Thus, at any time the current contents of the profiler stack reflect all profiler tags whose code is currently being executed. Profiler stacks are represented in the code by instances of `gw.api.profiler.ProfilerStack`.

If a profiler stack has been initialized for the current thread, the call to `Profiler.push(ProfilerTag.MYTAG)` pushes a new frame with tag `MYTAG` on to that profiler stack. Otherwise, the call has no effect.

Similarly, `Profiler.pop(frame)` is just a pass-through to calling `pop()` on the profiler stack of the current thread.

Properties and Counters on a Frame

Profiler frames can hold user-defined properties and counters that provide more information about system events. Consider this example:

```
frame.setPropertyValue("PARAMETER", str);
int ret = myMethod(str);
frame.setCounterValue("RETURN", ret);
```

When the profiler frame is popped off the stack, the frame contains information about which parameter was passed to `myMethod()` and the return value. Currently, properties and counters are used, among other things, to record SQL being executed, parameters to that SQL, row count returned, which rule is being executed, and so forth.

> **Note:** Exercise care when using this feature. Storing too much information will cause the display to become too cluttered, require more space for storage and, for long-running processes, hold on to too much memory at runtime.

Entry Points

The following entry points can initiate work on the application server:

Entry point	Description
Web	A user clicks around in a browser.
Batch process	Batch processes wake up at regular intervals, and can execute large queries. See “Batch Processing” on page 99.
Work queue	Long running processes that pick up work to be done from a queue. These processes are typically distributed across several servers. See “Batch Processing” on page 99.
Message destination	The process that sends messages out. See “Messaging and Events” on page 289 in the *Integration Guide*.
Web service	SOAP requests received by PolicyCenter. See “Web Services” on page 35 in the *Integration Guide*.
Startable plugin	You can create a plugin that is initialized at server startup and deinitialized at server shut-down. For example, a startable plugin can be registered as a listener on a JMS queue. See “Startable Plugins Overview” on page 259 in the *Integration Guide*.

The Guidewire Profiler provides a unified means of configuration for profiling these entry points and also provides a unified way of accessing Guidewire Profiler data.

Configuring the Profiler

Configure the Guidewire Profiler on the **Configuration** tab of the Guidewire Profiler page.

You can choose to enable or disable profiling for specific entry points. Except for Web entry points, configuration information is stored in the database and is visible to all application servers in the cluster. Note that any changes will take some time to propagate through the cluster and it may take up to the cache stale time for a change to become visible.

The next time the entry point is started, the Guidewire Profiler checks whether profiling is enabled for the entry point. If profiling is enabled, profiling data is recorded in the form of a profiler stack. Multiple stacks are recorded if the initial thread spawns more and the developer profiles the spawned threads. Except for Web, this data is persisted to the database and can be later retrieved.

For Web entry points, profiling can only be enabled for the current session. On the **Configuration** tab, click **Enable Web Profiling for this Session** to enable profiling for the current session. All subsequent round-trips to the server will be recorded as a separate profiler stack. As opposed to stacks from other types of entry points, stacks from Web requests are not persisted to the database. Instead, stacks from Web requests are stored in the user session. Guidewire recommends that you enable the Web profiler only when it is needed. Enable the profiler, exercise the pages that you want to profile and then disable the profiler. Furthermore, log out after the profiler data has been analyzed to free memory used by the profiler.

Additional Tracing

You can enable additional tracing options. These options are quite expensive to compute. Guidewire recommends that you narrow down your performance issues first before trying these options.

Tracing option	Description
Individual Stacks	Only available for work queue entry points. When checked, this stores one stack per work item and does not roll up the data. Use caution with the **Individual Stacks** option as the amount of data to store could be unbounded. The **Individual Stacks** option is recommended for use when there are only a few items in the queue.
Stack Trace Tracking	Captures the Java stack trace and the PCF trace at the point where a query is executed.
Query Optimizer Tracing	Oracle only. This trace indicates how the database arrived at a specific execution path. This creates a trace file on the database server.
Extended Query Tracing	Oracle only. This trace tracks the entire parse-execute-fetch of an SQL statement including what it is waiting on, if anything. This creates a trace file on the database server.
Diff DBMS Instrumentation Counters	Oracle only. Enable this option to capture the DBMS counters at the beginning of the profiling session and to include analysis of the differences in the DBMS-specific report.
DBMS Instrumentation Capture Threshold for each Action (millis)	Oracle only. The profiler generates a DBMS report if an action exceeds the threshold value set by this option. Set this threshold after you enable **Diff DBMS Instrumentation Counters**.

ProfilerAPI

You can use the `ProfilerAPI` web service to configure the profiler from an external system. In addition to enabling and disabling profiling for the various entry points, you can enable Web profiling on all subsequent sessions. This API does not provide the ability to profile a specific session or to profile active sessions. See “Profiling Web Services” on page 99 in the *Integration Guide*.

Analyzing Profiler Data

The Guidewire Profiler provides the following views of profiler data.

View Type	Result
Stack Queries	Lists the queries that were executed by each stack. The first list shows the stacks in the profiled session. The second shows the queries executed in that stack. More details can be obtained by clicking on each tab.
Aggregated Queries	Lists all queries executed as part of the profiled session and some statistics about them, including number of times executed, average time, and more.
Search by Query	Searches the session for a query. This view enables you to determine the source of a particular query. Paste in a query, for example from the AWR report, and click **Search**.
Elapsed	Lists each frame in chronological order within its stack along with the time in seconds between when PolicyCenter pushed and then popped the frame.
Chrono	Lists each frame in chronological order within its stack along with the time in seconds between PolicyCenter creating the stack and pushing the frame.
Group Frames	Lists frames in each stack aggregated by tag and presented in order of total aggregate time on the stack.
Group Stacks	Similar to **Group Frames**, except the Profiler aggregates the frames across all stacks in the session instead of by stack.
Rule Execution	Lists rules that fired during the session. If no rules fired during the session, the **Profiler Result** pane contains the message “No profiler stacks found”. See “Generating a Rule Execution Report” on page 48 in the *Rules Guide*.

Downloading and Uploading Profiler Data

PolicyCenter serializes profiler data and stores it in the database in a different table for each entry point. You can download and upload profiler data as a ZIP file. To download profiler data, enable profiling for a particular entry point, select the **Profiler Analysis** tab and then select the entry point. Then, click **Download.**

To upload a ZIP file of profiler data, select **Profiler Analysis** → **Saved File.** Then click **Upload.**

Profiling Custom Code

You can profile your custom code by using the Guidewire Profiler.

Creating New Profiler Tags

Define a new profiler tag to associate with the code that you want to profile. To do this, create a globally-accessible Gosu class that extends `gw.api.profiler.PCProfilerTag`. Within this class, define all of your custom profiler tags. Define these tags as constants. For example:

```
static final ProfilerTag MYTAG = new ProfilerTag("MyTag");
```

You use this class to describe the piece of code that you want to profile. put all their profiler tags as constants (static final) in one globally-accessible class.

To profile a block of your custom code, use the following pattern to push and pop profiling information onto the profiler stack. This code works for both Gosu and Java.

```
ProfilerFrame frame = Profiler.push(ProfilerTag.MYTAG);
try {
  /// CODE YOU WANT TO PROFILE
} finally {
Profiler.pop(frame);
}
```

Profiling spawned threads

Some processes spread their workload across multiple threads. If you want to profile those threads, use the following pattern:

```
gw.api.profiler.Profiler.createPotentiallyProfiledRunnable(ProfilerTag entryPointTag,
                                                           String entryPointDetail, GWRunnable block)
```

This generates a new `Runnable` that executes the given block. This `Runnable` profiles the block if the calling thread is also being profiled. The stack for that thread is associated with the stack of the calling thread and persisted along with the stack of the calling thread. See the javadoc for the `createPotentiallyProfiledRunnable` method for more details.

Cache Info

The **Cache Info** page provides information about the application server cache usage for PolicyCenter. Use this page to review how well the cache is performing. The **Cache Info** page provides graphical representations of the application server cache, in **Cache Summary**, **Historical Performance** and **Cache Details** views. You can refresh any of these views by clicking **Refresh**.

The **Cache Summary** tab graphs include:

Graph	Description
Cache Size	The memory used by the cache over time.
Hits and misses (Stacked)	The number of cache hits (an object was found in the cache) and misses (object was not found in the cache) and the miss percentage.
Type of Cache Misses	The number of cache misses caused by PolicyCenter evicting an object because the cache was full and the number of missed caused by PolicyCenter evicting an object due to reaping.
Evict Information	Information about cache evictions over time, including: • Number of times no entry was found to evict when cache was full • Number of evictions within active time when cache was full • Number of evictions when cache was full • Number of evictions due to reaping
Current Age Distribution	The number of objects of various ages in the cache.
Current Cache Contents for age All	The percentage of types of objects present in the cache for all ages.

If the `GlobalCacheDetailedStats` parameter in `config.xml` is set to `false`, the **Cache Info** page does not include the **Evict Information** and **Type of Cache Misses** graphs.

Click **Edit** to modify the maximum cache space and stale time parameters from the **Cache Summary** tab. If you change these parameters from the **Cache Summary** tab, the values you specify only apply to the application server node to which you are connecting and do not persist. If you restart the server, your changes are lost. To change these parameters and have the changes persist, edit the `config.xml` file. See “Application Server Caching” on page 65.

Click **Download** to download a CSV file containing detailed cache information.

Click **Clear Global Cache** to clear the cache entirely of all entities. The cache always contains some objects to support an active application server.

The **Historical Performance** tab graphs include:

Graph	Description
Space Retained	Memory used by the cache over the past couple days. The time shown is a much longer period than the cache size graph on the **Cache Summary** tab, which only displays the past 15 minutes.
Hits and Misses (stacked)	Number of hits (object was found in the cache) and misses (object was not found in the cache) and the miss percentage over the past day and past seven days.
Miss %	The percentage of cache read attempts in which the object was not found in the cache over the past day and the past seven days.
Number of Misses because item was evicted when cache was full	The number of misses over the past day and the past seven days due to PolicyCenter having evicted an object from the cache because the cache was full.

The **Cache Details** tab graphs include:

Graph	Description
Age Distribution by time	A number of graphs that show the age distribution of objects in the cache. The **Cache Details** tab shows age distributions for zero to 30 minutes ago.
Current Cache Contents by age	A number of graphs that show the percentage of types of objects in the cache over time.

Startable Plugin

The **Startable Plugin** page lists startable plugins that you have registered and enables you to manually start each plugin. For more information about startable plugins, see “Startable Plugins Overview” on page 259 in the *Integration Guide*.

Set Log Level

Use the **Set Log Level** option to set the logging level for different logging categories. The logging level you specify persists until you change it or restart the server.

The logging categories available depend on your integrations and the settings in the `logging.properties` file. Access this file from Guidewire Studio at **configuration** → **config** → **logging**.

Each logging category has an associated level. You can change the runtime **Levels** value for each category by using this dialog. If you restart the server, PolicyCenter does not retain your runtime settings. To make settings permanent, edit the `logging.properties` file. See “Configuring Logging” on page 23 for more information about how to configure logging and a description of the default logging categories.

View Logs

From the **View Logs** page you can view a log file, filter the log file for specific entries, and set the maximum number of lines to display.

To specify the location where the **View Logs** page checks for log files, specify the `guidewire.logDirectory` property in `logging.properties`. See “Specifying Location of Log Files for the View Logs Page” on page 24.

By default, PolicyCenter writes log files to `tmp/gwlogs/PolicyCenter/logs/`. You can configure log file locations and other properties for log files in the `logging.properties` file. See “Configuring Logging” on page 23.

Info Pages

The **Info Pages** provide information to help manage a PolicyCenter server and database. Guidewire intends these pages for use by Guidewire Support, Integration Engineers, Database Administrators, and System Administrators to diagnose existing and potential database-related performance problems. You can also use these pages to review the results of a load operation. You can access the following **Info Pages**:

- **Archive Info**
- **Configuration**
- **Domain Graph Info**
- **Consistency Checks**
- **Database Table Info**
- **Database Parameters**
- **Database Storage**
- **Data Distribution**
- **Database Statistics**
- **Oracle Statspack**
- **Oracle AWR**
- **Oracle AWR Unused Indexes Information**
- **SQL Server DMV Snapshot**
- **Microsoft JDBC Driver Logging**
- **Load History**
- **Load Integrity Checks**
- **Load Errors**
- **Upgrade Info**
- **Update Statistics**
- **Runtime Environment Info**
- **Safe Persisting Order**
- **Loaded Gosu Classes**

Archive Info

Use the **Archive Info** page to view information about the archive. You must have the `ArchiveEnabled` parameter set to `true` in `config.xml` to view the **Archive Info** page.

The **Archive Info** page includes an overview, information about the archiving plugin, a summary of archiving information by data model version and details of each Archive work queue run.

Click **Refresh** to update the information on the **Archive Info** page. The **Refresh** button also refreshes the `IArchiveSource` plugin.

Click **Download** to download the archive information to an HTML report. This report includes the information shown on the **Archive Info** page.

Click **View Progress** to open the **Work Queue Info** page so that you can view the progress of the Archive work queue. To start an unscheduled run of the `archive` work queue, navigate to the **Work Queue Info** page. Then click the **Run Writer** button for the **Archive** work queue. For more information, see “Work Queue Info” on page 135.

The **Overview** includes the number of entities that have been archived and the number that have been skipped or excluded. Entities excluded from archiving are divided into those entities excluded due to business logic and those excluded due to a failure. Click **Reset** to reset the total number of entities excluded. Details for each skipped and excluded entity are provided at the bottom of the **Archive Info** page.

The **Archive Source Information** indicates when the **Archive Info** page and `IArchiveSource` plugin were last refreshed. To update the **Archive Info** page, click **Refresh**. The **Archive Source Information** also shows the availability of the store, retrieve and delete services. These are based on the `storeStatus`, `retrieveStatus` and `deleteStatus` of the `ArchiveSource.gs` plugin. Possible values for these services include:

- **Available** – The service is available.
- **Failure** – The last attempt to archive, restore or delete failed.
- **Manually** – The service has been manually flagged as unavailable.
- **Not configured** – The service has not been configured.
- **Not enabled** – Archiving has not been enabled.
- **Not started** – Archiving has not yet been started.
- **Queue** – The service is not available but allow queueing of user requests.

The **Archive Summary by Datamodel Version** shows archiving information by data model version. This information includes the earliest date and the latest date that entities were archived with each data model version. For each version the **Archive Info** page shows the number of entities that were archived, the number excluded due to business logic, and the number excluded due to a failure. Each excluded category has a **Reset** button to reset the count of excluded entities. Click a data model version to see the **Archive Summary** page for that data model version. This page includes the **Reason for exclusion** for entities excludes by business logic and entities excluded because of failure. For each reason, you can click **Reset All Items** to reset the count. On the **Archive Summary** page, you can specify a **Begin Time** and **End Time** to limit the information shown for the data model to a specific date and time range.

Note: All archiving and restoring operations also produce the usual log information.

See also

- "More Information on Archiving" on page 325 in the *Application Guide* for a list of topics related to archiving.

IMPORTANT Guidewire strongly recommends that you contact Customer Support before implementing archiving.

Configuration

The **Configuration** page lists configuration parameters for your PolicyCenter environment. This page also includes a **Download** button. Click **Download** to download a copy of the following configuration files:

- `config.xml`
- `messaging-config.xml`
- `scheduler-config.xml`
- `work-queue.xml`

These files are located in the `config` folder within the downloaded ZIP file. The ZIP file also includes a `current` directory, which includes the in-memory state of `config.xml` and `work-queue.xml` parameters on the server. The in-memory state is made available because certain configuration parameters can be changed using a web service or JMX APIs after server startup.

Domain Graph Info

The **Domain Graph Info** page includes a **Graphs** tab that provides a DOT format representation of the domain graph and a **Warnings** tab to report issues with the graph. The DOT format is a means of showing object relationships in plain text. You can download the DOT format files and use a third-party tool, such as Graphviz, to view the graphs in a diagram format.

The Domain Graph

The domain graph represents the set of entities that relate to a root entity. The root entity is the main entity in the domain graph.

The domain graph defines the unit of work for object archiving. The unit of work for the archive process is a single instance of the graph. It is possible to associate certain entities with multiple graphs.

See also

- "The Archiving Domain Graph" on page 247 in the *Configuration Guide*
- "Archiving and the Domain Graph" on page 445 in the *Configuration Guide*

Viewing the Graphs

If you view a graph as a diagram:

- The direction of the arrows in the diagram shows the direction of the "is owned by" relationship. Most of the time, this is also the direction of the foreign key. If the relationship is in the opposite direction to the foreign key, then the edge between the two entities is drawn in blue.
- A bold arrow from an entity to itself represents an edge foreign key.
- An open arrow from the array entity represents arrays and one-to-one relationships.
- Blue entities and links indicate extensions.

Warnings

The **Warnings** tab shows any violations of the following warning-level checks.

PolicyCenter provides warnings for these situations rather than preventing the server from starting because business logic may prevent the erroneous situation. The server also performs other graph checks while starting. If these checks fail, the server does not start. Because the server does not start, you cannot use the **Archive Graph Info** page to view errors detected by these checks. Instead, the server reports the error and prints the graphs in DOT notation. You can use this output with a graph visualization program to view the graphs. For more information, see "Domain Graph Validation" on page 255 in the *Configuration Guide*.

Check	Description
Domain graph entities refer only to administrative data and domain entities	All foreign keys from entities in the domain graph only reference other entities in the domain graph. Otherwise, foreign key violations can occur as PolicyCenter traverses the domain graph during archiving processes.
Nothing outside the domain graph points to the domain graph	There must not be foreign keys from entities outside of the domain graph to entities in the domain graph. This prevents foreign key violations as PolicyCenter traverses the domain graph. This is not an outright failure because the archiving rule may prevent archival of such graphs anyway.
Null links cannot make node unreachable	PolicyCenter constructs the domain graph by looking at foreign keys, but the graph might not be a connected graph if a nullable foreign key is null. If enough links are null, the graph would become partitioned and the archiving or purging process would not be able to tag the correct entities. This check is a warning rather than one that prevents the server from starting because business logic might be in place that prevents the issue.

Download

Click **Download** to download information from the **Domain Graph Info** page. The downloaded ZIP file includes the following:

- `index.html` - a page that includes links to `construction.log`, `domain.dot` and `admin.dot` and shows any graph check warnings reported by PolicyCenter.
- `domain.dot` - a text file containing the domain graph in DOT format.
- `construction.log` - a text file that includes a log of how PolicyCenter constructed the graphs.
- `sorttable.js` - a javascript file containing javascript libraries used by `index.html` for showing graph warning information.

Consistency Checks

The **Consistency Checks** page enables you to view and run consistency checks on the PolicyCenter database.

The **View consistency checks definitions** tab lists the consistency checks available for each database table and provides a description of each check. You can search for consistency check types to see a list of all tables for which that consistency check is available. You can also search by table name to find the consistency checks related to that table. Most consistency checks operate on the specified table, but some checks, such as typelist table checks, operate on other tables. You can click **Download All** to download a ZIP file that contains HTML files describing all of the consistency checks provided with PolicyCenter. To view the downloaded information, extract the ZIP file and open the `index.html` file.

From the `index.html` file or the **View consistency checks definitions** tab you can do the following:

- Click **Table Name** to sort consistency checks by table name.
- Click **Check Name** to sort consistency checks by check name.
- Select the **Command** tab to view the SQL command of the consistency check. The SQL command retrieves a count of rows that violate the consistency check.
- Select the **Query to identify rows** tab to view the SQL query used to identify rows that violate the consistency check. SQL queries to identify rows that violate consistency checks are not available for all check types.

From the `index.html` file only, you can do the following:

- Click a table name to view all consistency checks related to that table.
- Click a check name to view all tables that the consistency check runs against.

You can run consistency checks from the **Run consistency checks** tab of the **Consistency Checks** info page. You can specify to run consistency checks for all tables, for specific tables, or for a defined table group. To define table groups, see “Defining Table Groups” on page 69 in the *Installation Guide*. You can specify to run all checks or only specific consistency check types. You must specify one or more tables and one or more check types. When running consistency checks from the **Run consistency checks** tab, you can specify a **Description**. PolicyCenter prepends the description when you view the results to a standard description of the tables and checks.

To see which consistency check types are available for each table, open the **View consistency checks definitions** tab of the **Info Pages** → **Consistency Checks** page. Enter the **Table name fragment** and click **Search**. PolicyCenter lists the available consistency checks, by name, for the table.

The **Run consistency checks** tab also shows the results of prior consistency check runs. You can download the results of past consistency check runs. If there are consistency check errors, the results include SQL queries that you can use to identify records that violated the consistency check.

You can also run consistency checks with the `system_tools -checkdbconsistency` command. Guidewire provides this command so that you can schedule consistency checks to run asynchronously when the database is handling fewer requests. See “Running Consistency Checks with System Tools” on page 40 and “System Tools Command” on page 162.

Database Table Info

Use the **Database Table Info** page to locate index and key information for each table. These pages document the names of the PolicyCenter generated indexes and constraints. The following tabs appear on this page:

Tab	Description
Indexes by Table	Lists the indexes on a table and provides information about the key columns associated with it.
Primary Key Constraints by Table	Lists the primary key constraint on a table and provides information about the fields that reference the key.

Tab	Description
Foreign Key Constraints by Table	Lists the foreign key constraints on a table and provides information about the table referenced by the key.
Number of columns and min/max row lengths	Displays the number of columns and the minimum and maximum row length in each table. Overly large row lengths in a database can lead to inefficiencies in data queries.

If your database is receiving integrity check errors or referential integrity problems, Guidewire Support might ask you to download the information from this page and provide it to them.

You can also click Verify to have PolicyCenter compare the database schema with the schema defined in the data model files.

Database Parameters

The **Database Parameters** page displays information about the database configuration. A drop-down menu provides a list of database parameter types that you can view on this page. The following selections are available:

Tab	Description
Database and Driver	The versions of the database and its associated driver.
Database Connection Pool Settings	Connection pool settings as configured in `config.xml` if using PolicyCenter to manage the connection pool. See “Configuring Connection Pool Parameters” on page 37 for more information on these parameters. If you use the application server to manage the connection pool, then this page does not show connection pool parameters. Instead, tune the connection pool by using the Administrative Console of the application server.
Guidewire Database Config	Guidewire-specific database configuration parameters. PolicyCenter reads these parameters from the `database` block in `config.xml` or uses a default value if `config.xml` does not specify database parameters.
Guidewire Database Config Statistics Settings	Guidewire-specific database configuration parameters related to statistics gathering.
Guidewire Database Upgrade Configuration	Guidewire-specific database configuration parameters related to upgrade.
Database Connection Properties	Properties related to the database connection, including whether **Autocommit** is on, the **Transaction isolation level** and whether the database connection is **Read Only**. This does not include the JDBC URL or credentials information. PolicyCenter shows the JDBC URL under **Database Connection Pool Settings**.
SQL Server Server Global Server Settings	SQL Server only. This view describes global server settings for the SQL Server instance.
SQL Server Database Options	SQL Server only. This view shows options set on the SQL Server database, such as `auto create statistics` and `auto update statistics`, the recovery model, collation, and so forth.
SQL Server Server Instance Attributes and Values	SQL Server only. This view shows attributes and values for the SQL Server instance to which PolicyCenter is connected.
SQL Server Session Properties	SQL Server only. This view shows properties of the SQL Server session for the current connection with PolicyCenter.

You can click **Download Database Parameters Info** to download an HTML file containing all database parameters.

If you troubleshoot a database performance issue with Guidewire Support, Guidewire Support might ask you to send this parameter information as a reference. Integration consultants also use this data while tuning database performance.

Database Storage

The **Database Storage** page provides information about the space and memory taken up by the database on the server. You can view and download database storage information.

The following tabs appear on this page:

Tab	Description
Database Space	Details about the amount of disk spaced taken up by the database.
Data Spaces	Lists the tablespaces (Oracle) or filegroups (SQL Server) taken up by the data and the amount of space taken and allocated by PolicyCenter.
TempDB Summary	Shows paging information for the database.
Indexes with High Fragmentation	Display average percentage of fragmentation for indexes in the database.
Index Physical Statistics	Lists the statistics about indexes on the physical table. Includes information such as minimum, average, and maximum record size. To change which **Tables and Indexes** the **Index Physical Statistics** tab displays, select a new one and click **Refresh**.
Tables and Indexes	Lists paging and allocation type of a table and its indexes. To change which table the **Tables and Indexes** tab displays, select a new table and click **Refresh**.

Download and save the database storage information right before and after an upgrade or other significant database change. This provides you with a point of reference you can provide to Guidewire Support if requested.

Data Distribution

Use the **Data Distribution** page to start a database distribution batch job. After the batch job completes, you can click **Refresh**. PolicyCenter lists the available distribution reports for download. PolicyCenter maintains each generated report until you **Delete** it.

Click **Download** to download a ZIP file containing the reports. The ZIP file contains HTML pages with the generated data.

To view the reports

1. Unzip the file into its own directory.
2. Locate the `index.html` file and double-click it to open it in a browser.
3. Use the links on the page to navigate through the distribution reports.

This page also lists data distribution reports for processes you start from the command line. To start the batch process from the command line, enter the following:

```
maintenance_tools -password password -startprocess datadistribution
```

Database Statistics

The **Database Statistics** page provides reports about out-of-date statistics in the database indexes, histograms, staging tables, and PolicyCenter tables. For information about generating the database statistics, see “Configuring Database Statistics” on page 42.

On the **DatabaseStatisticsInfo** tab of the **Database Statistics** page, you can view database statistics reports for the entire database or for specific tables. To specify tables on which to gather statistics, select **No** for the **View database catalog statistics on all tables** option. Then select the checkbox next to each table for which you want statistics. You can also not select any tables and PolicyCenter reports statistics information only from the database metadata.

Click **Download** to download the statistics report.

PolicyCenter database statistics reports include the following information.

Tab	Description
Unanalyzed Indexes	Lists the indexes that were not broken out for statistical purposes.
Unanalyzed Histograms	Lists the histograms that were not broken out for statistical purposes.
Stale Index Stats	Details potentially out-of-date indexes on PolicyCenter tables. PolicyCenter considers the statistics of an index as potentially out-of-date if the estimated row count does not match the actual row count. However, it is not uncommon for the estimates to differ from the actual numbers without there being a problem. To confirm if the statistics are out-of-date, you or your company's DBA must perform an analysis.
Stale Histogram Stats	Details potentially out-of-date histograms on PolicyCenter tables. The report considers histogram statistics as potentially out-of-date if the estimated row count does not match the actual row count. However, it is not uncommon for the estimates to differ from the actual numbers without there being a problem. To confirm if the statistics are out-of-date, you or your company's DBA must perform an analysis.
Application Tables	Displays statistics for individual PolicyCenter tables. Use the **Pick table to display stats** drop-down to select a table. The drop-down lists the current row count for each table. After you select the table, you can view the data associated with indexes and histograms (if defined) on the table.
Staging Tables	Displays statistics for individual staging tables. Use the **Pick table to display stats** drop-down to select a table. The drop-down lists the current row count for each table. After you select the table, you can view the data associated with indexes and histograms (if defined) on the table.
Typelist Tables	Displays statistics for individual typelist tables. Use the **Pick table to display stats** drop-down to select a table. The drop-down lists the current row count for each table. After you select the table, you can view the data associated with indexes and histograms (if defined) on the table.

PolicyCenter provides database statistics generation designed specifically for how the PolicyCenter application and data model interact with the physical database. Generating database statistics from the database management system can potentially create statistics that cause PolicyCenter to select a bad plan for execution of SQL queries against the database. Therefore, always use PolicyCenter to generate database statistics, rather than by using the statistics generation provided with the database management system.

The **Execution History** tab of the **Database Statistics** page lists when the command to update database statics has been run. In development mode, you can also generate full or incremental database statistics directly from the **Execution History** tab of the **Database Statistics** page.

Oracle Statspack

This page is available only if the database server is Oracle. To display statspack information, you must have installed the statspack option in your Oracle database. Refer to Oracle documentation for instructions. You must also create statspack snapshots by using a tool such as SQL*Plus or SQL Developer. The **Oracle Statspack** page displays statspack snapshots you have created. A snapshot gives you database configuration and performance statistics for the duration you defined while creating the snapshot.

Oracle AWR

This page is available only if the database server is Oracle. The Oracle Automatic Workload Repository (AWR) is an upgrade of the information available with the Oracle statspack. Refer to Oracle documentation for details. You can use the **Oracle AWR** page to select AWR snapshots you define in the database and generate a performance report using those two snapshots. The snapshots must have the same Oracle instance startup time.

You can also retrieve performance reports based on Oracle AWR snapshots from the command line using the `system_tools`:

```
system_tools -user user -password password -oraPerfReport beginSnapshotID endSnapshotID
  probeVDollarTables
```

Specify the beginning and ending snapshot IDs and whether to probe V$ tables. The two snapshots must share the same Oracle instance startup time.

The third argument can also specify a file by prefixing the filename with an @ sign:

```
system_tools -user user -password password -oraPerfReport beginSnapshotID endSnapshotID
  @filename.properties
```

The filename can optionally be prefixed with the path to the file if not in the current directory. This file is a standard properties file with the following property names (default value in parenthesis):

- probleVDollarTables (false)
- capturePeekedBindVariables (false)
- searchQueriesMultipleHistoricPlans (false)
- searchQueriesBeginSnapOnly (true)
- searchQueriesEndSnapOnly (true)
- includeInstrumentationMetadata (false)
- outputRawData (false)
- includeDatabaseStatistics (true)
- probeSqlMonitor (true)
- capturePeakedBindVariablesFromAWR (false)
- genCallsToAshScripts (false)

Each property must be spelled and capitalized as shown or it is ignored. Each property, if specified, must be set to `true` or `false`. If the property is not specified, the default value is taken for the property. If you generate a performance report from the **Oracle AWR** page, you can also specify these options on the page.

To retrieve a list of recent Oracle AWR snapshot IDs to choose from, use the following command:

```
system_tools -user user -password password -oraListSnaps numSnaps
```

The `system_tools -oraPerfReport` command reports the process ID of the process generating the performance report. You can check on the status of this process using the following command:

```
maintenance_tools -user user -password password -processstatus processID
```

See “System Tools Command” on page 162.

Oracle AWR Unused Indexes Information

This page is available only if the database server is Oracle. The **Oracle AWR Unused Indexes Information** page enables you to select two snapshots from the same instance startup time. Select snapshots that have a wide range. The downloadable report provides information about indexes that have no logical or physical reads or are not found in query plans.

SQL Server DMV Snapshot

The **SQL Server DMV Snapshot** page is available only if the database server is SQL Server. Use this page to generate and download performance reports using SQL Server Dynamic Management Views. You can specify the number of **Top Queries to Report**, **Hot Objects to Report**, and whether to **Include Database Statistics** in the report. Click **Generate Perf Report** to launch an internal batch process that gathers performance data and creates the report. Performance reports are listed in a table with **Download** and **Delete** buttons for each report.

You can also generate a DMV report using a batch process from the command line:

```
system_tools -user user -password password -mssqlPerfRpt numTopQueries numHotObjects collectStatistics
```

Replace *numTopQueries* and *numHotObjects* with integer values for the number of top queries and hot objects to report. Replace *collectStatistics* with `true` or `false` to specify whether PolicyCenter gathers database statistics while generating the DMV report. If you do not specify any parameters, PolicyCenter uses defaults of 400 top queries, 400 hot objects, and collects statistics. You can specify either zero or all three parameters only.

See “System Tools Command” on page 162.

Microsoft JDBC Driver Logging

The **Microsoft JDBC Driver Logging** page is available only if the database server is SQL Server. Use this page to set the **Logging Level** for the Microsoft JDBC driver. Be cautious setting the logging level, as detailed logging can slow the system significantly. You can specify the **Logging Format** as **Simple**, for a more readable format, or **XML** for XML output, usually parsed by another system. You can also specify the **Log File Location**, including special components that are replaced at runtime, such as `%u` to append a unique number to each log to avoid conflicts.

Load History

The **Load History** page displays information about database load operations that completed in PolicyCenter. These load operations execute as you import data into the database. For example, loading data into the staging tables impacts the loader history information.

This page contains a summary view and a detail view. The summary view lists information about each specific load operation such as who called it, when, how long the operation took, any errors generated, and so forth. You can drill down into the details of an operation by clicking **View**.

The detail view shows on two tabs the **Steps** in the operation and the impact of the operation on **Row Counts**. You can drill down into the individual **Steps** by clicking on the step. Use the **Row Counts** page to quickly assess whether the amount of data that the operation loaded was the amount that you expected the operation to load.

See “Zone Import” on page 541 in the *Integration Guide*.

Load Integrity Checks

The **Load Integrity Checks** page reports on the SQL integrity checks that run as a database load operation executes. This page has two tabs: **View by Staging Table** and **View by Load Error Type**. To view the checks by table, select the former. To filter by error type, choose the latter.

For each integrity check, the **Load Integrity Checks** page lists the SQL query that the check performs. The **Load Integrity Checks** page also lists a description of the check and the associated load error type or staging table.

On both tabs, you can enable **Allow Non Admin References** or not. If you allow them, then PolicyCenter checks foreign key references to administrative tables (such as users and groups) on load. If this value is `false`, PolicyCenter does not check these references. The value is `false` by default.

See “Data Integrity Checks” on page 550 in the *Integration Guide*.

Load Errors

The **Load Errors** page displays errors generated by failed integrity checks. You can use this page to drill down through a table name to the specific error generated by a load operation. Errors relate to a particular staging table row. For each error, the **Load Errors** page shows:

- The table
- The row number
- The logical unit of work ID (LUWID)
- The error message
- The data integrity check query that failed.

In some cases, PolicyCenter cannot identify or store a single LUWID for the error.

See “Data Integrity Checks” on page 550 in the *Integration Guide*.

Upgrade Info

The **Upgrade Info** page displays information about the automatic upgrade that runs upon server startup. You can use this page to see what steps were run by the upgrade and the impacts to row counts and storage information. From this page, you can also see the database parameters used during the upgrade.

See also

- "Understanding and Authorizing Data Model Updates" on page 39
- "Viewing Detailed Database Upgrade Information" on page 189 in the *Upgrade Guide*

Update Statistics

The **Update Statistics** page provides downloadable ZIP files containing the results of update statistics operations. See "Configuring Database Statistics" on page 42 and "Database Statistics" on page 149.

Runtime Environment Info

The **Runtime Environment Info** page lists information about the runtime environment for PolicyCenter. This information includes Guidewire platform and PolicyCenter build information, system properties, and environment variables.

Safe Persisting Order

The **Safe Persisting Order** page lists the order in which PolicyCenter runs the Preupdate rules for their root entities.

Loaded Gosu Classes

The **Loaded Gosu Classes** page provides a list of all the Gosu classes that have been loaded by PolicyCenter.

Cluster Info

The **Cluster Info** page provides information on the clustered environment, if you have enabled clustering. The **Cluster Info** page provides the host name and server ID of the application server instance, the batch server, and each member of the cluster. Click **Refresh Cluster Info** to update the information on the page.

Product Model Info

The **Product Model Info** screen contains a single **Reload Availability** button. When you click this button, PolicyCenter attempts to reload availability data from the directory specified in the `ExternalProductModelDirectory` parameter defined in `config.xml`. The server attempts to synchronize the lookup entities and the existing product model availability data with the XML files stored in the external lookup directory. If the reload is successful, PolicyCenter displays an informational message. If reload is not successful, PolicyCenter displays an error message. In either case, you can check the server log files for details of the reload operations or problems that occurred.

You can access the **Product Model Info** screen if the following are true:

- If the server is in development mode, the `EnableInternalDebugTools` parameter in `config.xml` must be `true`.
- You have the `View ProductModelInfo tools page` permission. The code for this permission is `toolsProductModelInfoview`. In the base configuration, only the **Superuser** role has this permission.

See also

- "Reloading Availability Data" on page 95 in the *Product Model Guide*
- "Reloading Availability Example" on page 97 in the *Product Model Guide*

Free-text Search

The **Free-text Search** page helps you manage the Guidewire Solr Extension, a full-text search engine, during development. The page is an alternative to the free-text batch load command. The **Free-text Search** page provides one operation to drop the indexes and another operation to load and index data. The free-text batch load command performs both operations in a single command.

During development, use The **Free-text Search** page instead of the free-text batch load command to avoid the installation and setup procedure required to use the command.

To access the **Free-text Search** page, you must run the PolicyCenter application in development mode. The application hides the page whenever you run the application in production mode.

The **Free-text Search** page provides the following buttons:

- **Sync Policy Index** – Provides the functionality similar to the free-text batch load command. It extracts policy data from the application database and sends it to the Guidewire Solr Extension for indexing. Click this button after you click the **Drop Policy Index** button.
- **Drop Policy Index** – Drops the policy indexes from the Guidewire Solr Extension. Click this button before you click the **Sync Policy Index** button.
- **Run Consistency Check** – Confirms that the policy index data in the Guidewire Solr Extension matches the policy data in the application database. Click this button after you click the **Sync Policy Index** button or after you run the free-text batch load command.

You can access the **Free-text Search** page if all the following are true:

- The server is in development mode.
- The `EnableInternalDebugTools` parameter in `config.xml` is `true`.
- The `FreeTextSearchEnabled` parameter in `config.xml` is `true`.

See also

- "Setting Up the Free-text Batch Load Command" on page 99 in the *Installation Guide*
- "Running the Free-text Batch Load Command" on page 175
- "Free-text Search Configuration" on page 341 in the *Configuration Guide*

Using the Internal Tools

WARNING Guidewire does not support the **Internal Tools**. Guidewire provides these tools for use during development only. Guidewire does not support the **Internal Tools** for production environments. Use these tools at your own risk.

The **Internal Tools** page is only available if the server is in development mode. You can put the server in development mode by setting the JVM parameter `-Dgw.server.mode=dev`. Users with the `internaltools` permission can then access the **Internal Tools** pages by pressing `ALT+SHIFT+T` and selecting the **Internal Tools** tab. The **Superuser** role has the `internaltools` permission by default. For more information about server modes, see "Server Modes and Run Levels" on page 58.

This topic includes:

Reload

The **Reload** page is useful while you develop a configuration. From this page you can reload key configuration files into a running PolicyCenter installation. You can choose from the following options:

Option	Description
Reload PCF Files	Verifies and reloads all PCF files. If there are errors in the PCF files, PolicyCenter writes the errors to the log.
Verify All PCF Files	Verifies the PCF files without reloading them.
Reload Web Templates	Reloads the entire PolicyCenter user interface including the `config/web/templates` directory.
Reload Workflow Engine	Reloads the Workflow engine.
Reload Display Names	Reloads label definitions only from the `display.properties` for the locale.

PC Sample Data

The **PC Sample Data** page is for loading sample data into PolicyCenter for development purposes only. Guidewire does not support this tool for a production environment. See "Installing Sample Data" on page 55 in the *Installation Guide* for instructions.

Testing System Clock

The system clock plugin, `TestingClock`, enables you to get and set the current system time in PolicyCenter. This non-production tool is useful during the testing phase. You can move the system time forward as necessary to determine if a process completes correctly. You can not set the system time to a time before the current time.

The system clock plugin is for testing purposes only. You can only adjust the system clock if the PolicyCenter server is in development or test mode.

The `TestingClock` plugin must be enabled and configured to use this tool.

To configure `TestingClock` and verify it is enabled

1. Start Guidewire Studio with the `gwpc studio` command from `PolicyCenter/bin`.
2. In Studio, expand **configuration** → **config** → **Plugins** → **registry** and open `ITestingClock.gwp`.
3. Specify the **Environment** and **Server** to implement the plugin for a particular environment and server. Or, leave **Environment** and **Server** blank to implement the plugin in the default environment.
4. Verify that the **Enabled** checkbox is selected.
5. Save your changes.

For more information on the system clock plugin, see "Testing Clock Plugin (Only For Non-Production Servers)" on page 254 in the *Integration Guide* for details.

chapter 12

PolicyCenter Administrative Commands

PolicyCenter includes a number of commands that help with administrative tasks on your PolicyCenter server. The `PolicyCenter/admin/bin` directory contains these commands.

This topic includes:

See also

- For commands that build PolicyCenter, see "Build Tools" on page 119 in the *Installation Guide*.

Administration Tools Overview

PolicyCenter provides a set of administrative tools you can use to control the server from the command line. Typically, these commands are meant to run on an administrator's workstation. The tools are all found in the `PolicyCenter/admin/bin` directory, unless otherwise noted.

There are `*.bat` and `*.sh` versions of each administration tool to support installations on Windows and UNIX systems, respectively. You can only use these tools if the PolicyCenter server is running.

Unless otherwise specified, details of each command are located in “PolicyCenter Administrative Commands” on page 157.

The following table summarizes what each tool does.

File	Description
`import_tools`	A set of utilities for loading data into PolicyCenter. For help on syntax and available utilities, use: `import_tools -help`
`maintenance_tools`	A set of utilities for performing maintenance operations on the server (for example, running escalation/exception rules, calculating statistics, and more.) For help on syntax and available utilities, use: `maintenance_tools –help`
`messaging_tools`	Provides a set of utilities for managing integration event messages (for example, retrying a message, skipping a message, purging the message table, and more).
`system_tools`	Provides a set of utilities for controlling the server (for example, pinging the server, bringing the server in and out of maintenance mode, updating database statistics, and more.) For help on syntax and available utilities, use: `system_tools –help`
`table_import`	Used for importing tables into the database.
`template_tools`	Helps in converting between versions of document templates.
`workflow_tools`	Manages workflows.
`zone_import`	Loads zone data from a file to a staging table.

Import Tools Command

```
import_tools -help
import_tools -password password [-server url] [-user user] [-ignore_all_errors]
             { { -import filename1, filename2 ... [{-dataset dataset | -output_csv filename |
             -output_xml filename}] | -D name=value |
             -privileges } [-ignore_all_errors] [-ignore_null_violations] }
```

The `import_tools` command imports new or updated data into existing tables in the PolicyCenter database. You can only import data for valid entities or their subtypes. PolicyCenter supports this command for importing administrative data but not for importing other data into PolicyCenter.

Note: PolicyCenter does not fire any events related to the data you add or modify through this command.

Import Tools Options

You can use any of the following options with the `import_tools` command. You must always supply the `-password` option.

Option	Description
`-D name=value`	Sets a system property.
`-dataset integer`	Specifies the *integer* representing the dataset to import. Datasets are ordered by inclusion. The smallest dataset is always numbered 0. Thus, dataset 0 is a subset of dataset 1, and dataset 1 is a subset of dataset 2, and so forth. To import all data, set this value to -1.
`-help`	Displays the command usage.
`-ignore_all_errors`	Causes the command to ignore any errors in a CSV file.
`-ignore_null_violations`	Causes the command to ignore violations of null constraints in import data.

Option	Description
`-import filename1, filename2, ...`	Imports administrative data from a comma-separated list of CSV or XML files.
`-output_csv filename`	If used with the `-import` option, outputs CSV to the specified file and then stops processing. PolicyCenter imports no data into the server. Use this command to convert XML input files to CSV.
`-output_xml filename`	If used with the `-import` option, outputs XML to specified file and then stops processing. PolicyCenter imports no data into the server. Use this command to convert CSV input files to XML.
`-password password`	Specifies the *password* to use to connect to the server. PolicyCenter requires the password.
`-privileges`	Rebuilds role privileges by deleting role privileges in the current database and importing the `roleprivileges.csv` file in `PolicyCenter/modules/configuration/config/import/gen`.
`-server url`	Specifies the PolicyCenter host server url.
`-user user`	Specifies the *user* to run the process.

Maintenance Tools Command

```
maintenance_tools -help
maintenance_tools -password password [-server url] [-user user] [-help] { -D name=value |
                  -processstatus process | -startprocess process | -terminateprocess process
                  -whenstats}
```

The `maintenance_tools` command starts, terminates, or gets the status of a PolicyCenter process.

See also

- For a list of processes that the `maintenance_tools` command can start, see "Scheduling Work Queues and Batch Processes" on page 107.

Maintenance Tools Options

You can use any of the following options with the `maintenance_tools` command. You must always supply the `-password` option.

Option	Description
`-help`	Displays the command usage.
`-password password`	Specifies the administrative password. PolicyCenter requires a password to launch the maintenance tools.
`-processstatus process`	Gets the status of a batch process. For the *process* value, specify a valid process name or a process ID. For work queues, this command returns the status of the writer process. It does not check whether there are remaining work items. It is possible for the process status to report as completed because the writer has completed adding items to the work queue, yet there are remaining unprocessed work items.
`-server url`	Specifies the PolicyCenter host server url.
`-startprocess process`	Starts a new batch process. For the *process* value, specify a valid process code. For a list of batch process codes, including work queue writer processes, see "Scheduling Work Queues and Batch Processes" on page 107

Option	Description
`-terminateprocess process`	Terminates a batch process. For the *process* value, specify a valid process name or a process ID. Single phase processes cannot be terminated using this command. Single phase processes run in a single transaction, so there is no convenient place to terminate the process. The following batch processes are single phase processes that cannot be terminated: • AggregateLimitCalculation • DashboardStats • DataDistribution • ExchangeRate • FinancialsCalculation • Statistics • Table Import
`-user user`	Specifies the *user* to run the process.
`-whenstats`	Reports the last time PolicyCenter calculated statistics on the server.

Messaging Tools Command

```
messaging_tools -password password [-server url] [-user user] [-config destinationID] [-purge date]
                [-restart destinationID wait retries initial backoff poll threads chunk]
                [resume destinationID] [-resync -destination destinationID -policy -policyID]
                [-retry messageID] [-retrydest destinationID]
                [-skip messageID] [-statistics destinationID] [-suspend destinationID]
```

The `messaging_tools` command lets you manage a message destination from the command line. To manage a message destination from the command line, you must know its destination ID. The person who creates the message destination assigns this ID.

Messaging Tools Options

You can use any of the following options with the `messaging_tools` command. You must always supply the `-password` option.

Option	Description
`-config -destination destinationID`	Returns the configuration for a destination.
`-destination destinationID`	Specifies a specific message destination for resynchronizing. Used with `-resync`. Specify the policy with `-policy`.
`-password password`	Specifies the administrative password. You must specify a *password*.
`-policy policyID`	The policy ID to resync. Used with the `-resync` and `-destination` arguments.
`-purge date`	Deletes completed messages that are older than a specified date. The purge tool deletes messages in `Acked`, `ErrorCleared`, `Skipped` or `ErrorRetried` state with send time before the specified date. The date format is `mm/dd/YYYY`. If the purge tool succeeds in removing these messages without error, it reports "`Message table purged`". Since the number and size of messages can be very large, periodically use this command to purge old messages to avoid the database from growing unnecessarily.

Option	Description
`-restart -destination destinationID -wait wait -retries retries -initial initial -backoff backoff -poll poll -threads threads -chunk chunk`	Restarts the messaging destination with new configuration settings. *destination* – The destination ID of the destination to suspend. *retries* – The number of automatic retries to attempt before suspending the messaging destination. *initial* – The amount of time in milliseconds after a retryable error to retry sending a message. *backoff* – The amount to increase the time between retries, specified as a multiplier of the time previously attempted. For example, if the last retry time attempted was 5 minutes, and *backoff* is set to 2, PolicyCenter attempts the next retry in 10 minutes. *pollinterval* – Each messaging destination pulls messages from the database (from the send queue) in batches of messages on the batch server. The application does not query again until *pollinterval* amount of time passes. After the current round of sending, the messaging destination sleeps for the reminder of the poll interval. If the current round of sending takes longer than the poll interval, then the thread does not sleep at all and continues to the next round of querying and sending. See “Message Ordering and Multi-Threaded Sending” on page 323 in the *Integration Guide* for details on how the polling interval works. If your performance issues primarily relate to many messages per primary object per destination, then the polling interval is the most important messaging performance setting. *threads* – To send messages associated with a primary object, PolicyCenter can create multiple sender threads for each messaging destination to distribute the workload. These are threads that actually call the messaging plugins to send the messages. Use the `threads` parameter to configure the number of sender threads for safe-ordered messages. This setting is ignored for non-safe-ordered messages, since those are always handled by one thread for each destination. If your performance issues primarily relate to many messages but few messages per claim for each destination, then this is the most important messaging performance setting. For more information, see “Message Ordering and Multi-Threaded Sending” on page 323 in the *Integration Guide*. *chunksize* – number of messages to read in a chunk. *wait* – the number of seconds to wait for the shutdown before forcing it.
`-resume -destination destinationID`	Resumes the operation of the specified message destination.
`-resync`	Resyncs a policy with specified ID against a specific message destination. Use `-destination` and `-policy` to specify the destination and policy.
`-retry messageID`	Attempts to resend a message that failed. The message must be a candidate for retrying. A message is a candidate if the error at the destination system was temporary and the message destination has no automatic retry mechanism. For instance, if the record was locked and refused the update, the message would be a candidate for retrying.
`-retrydest destinationID`	Retries all retryable messages for a message destination.
`-server url`	Specifies the PolicyCenter host server url.
`-skip messageid`	Skips a message with the specified ID. If you mark a message as skipped, then PolicyCenter stops trying to resend the message. After you skip a message, you can not retry it.
`-statistics destinationID`	Prints the statistics for the specified destination.
`-suspend destinationID`	Suspends a message destination. Use this command if the destination system is going to be shut down or to halt sending while PolicyCenter processes a daily batch file.
`-user user`	Specifies the *user* to run the process.

System Tools Command

```
system_tools -help
system_tools -password password [-server url] [-user user] [-help] { -D name=value |
             -checkdbconsistency [tableSelection checkTypeSelection] |
             -daemons | -getdbstatisticsstatements | getincrementaldbstatisticsstatements |
             -listPerfReports [number] | -loggercats | -maintenance |
             -mssqlPerfRpt [numTopQueries numHotObjects collectStatistics] -multiuser |
             -oraListSnaps | -oraPerfReport [] | -ping |
             -recalcchecksums | -reloadloggingconfig | -sessioninfo |
             -updateloggginglevel loggername logginglevel | -verifydbschema | -version }
```

System Tools Options

You can use any of the following options with the `system_tools` command. You must always supply the `-password` option.

Option	Description
`-cancelupdatestats`	Cancels the process that is updating database statistics.
`-checkdbconsistency`	Checks the consistency of data in the database. The `-checkdbconsistency` option runs consistency checks as an asynchronous batch process. PolicyCenter provides this option so that you can schedule consistency checks to run when the database server is handling fewer requests. To run consistency checks manually, use the **Consistency Checks Info Page**, described at "Consistency Checks" on page 147. This tool has two optional arguments: `-checkdbconsistency` *`tableSelection checkTypeSelection`* The *`tableSelection`* argument can be specified as: • `all` – Run consistency checks on all tables. • *`table name`* – The name of a single table on which to run checks. • `tg.`*`table group name`* – The name of a table group. Table groups are defined in the `database` element of `config.xml`. For more information, see "Defining Table Groups" on page 69 in the *Installation Guide*. • `@`*`file name`* – A file name with one or more valid table names or table group names entered in comma-separated values (CSV) format. Prefix table group names with `tg.`, such as `tg.MyTableGroup`. You can combine table groups and individual table names in the same file. The *`checkTypeSelection`* can be specified as: • `all` – Run all consistency checks on the specified tables. • *`check name`* – The typecode of a single consistency check to run. • `@`*`file name`* – A file name with one or more valid consistency check names entered in comma-separated values (CSV) format. If you specify one optional argument, you must specify both. For more information, see "Checking Database Consistency" on page 40.
`-daemons`	Sets the server to the `DAEMONS` run level. For information about functionality available at various run levels, see "Server Modes and Run Levels" on page 58.

Option	Description
`-dbcatstats`	Used with no arguments, returns a ZIP file of database catalog statistics info for all the tables in the database. `-dbcatstats <regularTables stagingTables typelistTables>` Used with three arguments, returns a ZIP file of database catalog statistics info for the specified tables. Each of the arguments can be specified as: • `all`/`none` – Select all/none tables of this type, or • `<table name>` – The name of a single table of this type, or • `@<file name>` – A file name with one or more valid table names of this type entered in comma-separated values (CSV) format. For example, `-dbcatstats none none all` would return database catalog statistics information for all the typelist tables.You must specify either no arguments or three arguments while launching this tool. This option designates You can specify the target destination for the database catalog statistics ZIP file by adding the `-filepath filepath` parameter. If you do not provide a path, PolicyCenter uses the current directory. This process can take a long time, and it is possible for the connection to time out. If the connection times out while running this command, try reducing the number of tables to gather statistics on at a time by using the arguments shown previously. For information about configuring database statistics generation, see “Configuring Database Statistics” on page 42.
`-getdbstatisticsstatements`	Gets the list of SQL statements to update database statistics. See “Configuring Database Statistics” on page 42.
`-getincrementaldbstatisticsstatements`	Gets the list of SQL statements to update database statistics for tables exceeding the change threshold. The change threshold is defined by the `incrementalupdatethresholdpercent` attribute of the `databasestatistics` element in `database-config.xml`. See “Configuring Database Statistics” on page 42.
`-getPerfReport ID`	Downloads the performance report with the specified *ID*. You can retrieve a list of available performance report IDs by running the `system_tools -listPerfReports` command.
`-getupdatestatsstate`	Returns the state of the process running the statistics update.
`-help`	Displays the command usage.
`-listPerfReports [number]`	Lists IDs and other information for available database performance reports. You can specify an optional integer to specify the number of available downloads to list, ordered starting with the most recent. If unspecified or `0`, all available downloads are listed. The list shows the ID of the report and the status, indicating if the performance report batch job succeeded, failed, or is still running. The list also includes the start and end times of the batch job and the description of the batch run. You can use the ID of the performance report to download the report with the `system_tools getPerfReport ID` command.
`-loggercats`	Displays the available logging categories.
`-maintenance`	Sets the server to the `MAINTENANCE` run level. For information about functionality available at various run levels, see “Server Modes and Run Levels” on page 58.

Option	Description
`-mssqlPerfRpt [numTopQueries numHotObjects collectStatistics]`	Generates a SQL Server DMV performance report using a batch process: `system_tools -user user -password password -mssqlPerfRpt numTopQueries numHotObjects collectStatistics` Replace *numTopQueries* and *numHotObjects* with integer values for the number of top queries and hot objects to report. Replace *collectStatistics* with `true` or `false` to specify whether PolicyCenter gathers database statistics while generating the DMV report. You can specify all three parameters or none. If you do not specify any parameters, PolicyCenter uses defaults of 400 top queries, 400 hot objects, and collects statistics.
`-multiuser`	Sets the server to the `MULTIUSER` run level. For information about functionality available at various run levels, see “Server Modes and Run Levels” on page 58.
`-oraListSnaps numSnaps`	Lists *numSnaps* number of available Oracle AWR snapshot IDs, starting with the most recent snapshot. You can generate performance reports using the `-oraPerfReport` command with these available beginning and ending snapshot IDs.

Option	Description
`-oraPerfReport` *`beginSnapshotID endSnapshotID probeVDollarTables`*	You can retrieve Oracle AWR performance reports from the command line using the `system_tools`: `system_tools -user` *`user`* `-password` *`password`* `-oraPerfReport` *`beginSnapshotID endSnapshotID probeVDollarTables`* Specify the beginning and ending snapshot IDs and whether to probe `VDollar` tables. The two snapshots must share the same Oracle instance startup time. The third argument can also specify a file by prefixing the filename with an `@` sign: `system_tools -user` *`user`* `-password` *`password`* `-oraPerfReport` *`beginSnapshotID endSnapshotID`* `@`*`filename`*`.properties` The filename can optionally be prefixed with the path to the file if not in the current directory. This file is a standard properties file with the following property names (default value in parenthesis): • `probleVDollarTables` (`false`) • `capturePeekedBindVariables` (`false`) • `searchQueriesMultipleHistoricPlans` (`false`) • `searchQueriesBeginSnapOnly` (`true`) • `searchQueriesEndSnapOnly` (`true`) • `includeInstrumentationMetadata` (`false`) • `outputRawData` (`false`) • `includeDatabaseStatistics` (`true`) • `probeSqlMonitor` (`true`) • `capturePeakedBindVariablesFromAWR` (`false`) • `genCallsToAshScripts` (`false`) Each property must be spelled and capitalized as shown or it is ignored. Each property, if specified, must be set to `true` or `false`. If the property is not specified, the default value is taken for the property. To retrieve a list or recent Oracle AWR snapshot IDs to choose from, use the following command: `system_tools -user` *`user`* `-password` *`password`* `-oraListSnaps` *`numSnaps`* The `-oraPerfReport` command reports the process ID of the process generating the performance report. You can check on the status of this process using the following command: `maintenance_tools -user` *`user`* `-password` *`password`* `-processstatus` *`processID`* View the performance report on the Info Page. See “Oracle AWR” on page 150.
`-password` *`password`*	Specifies the administrative password. You must specify a *`password`*.
`-ping`	Pings the server to check if its active. The returned message indicates the server run level. The possible responses are: • `MULTIUSER` • `DAEMONS` • `MAINTENANCE` • `STARTING` For information about functionality available at various run levels, see “Server Modes and Run Levels” on page 58.
`-recalcchecksums`	Recalculates file checksums used for clustered configuration verification.
`-reloadloggingconfig`	Directs the server to reload the logging configuration file.
`-server` *`url`*	Specifies the PolicyCenter host server url.
`-sessioninfo`	Returns the session information of the server.

Option	Description
`-updatelogginglevel logger level`	Sets the logging level of logger with the given name. For the root logger, specify `RootLogger` for the *`logger`* name.
`-updatestatistics description true|false`	Launches a process to update database statistics. Specify `true` to update database statistics only for tables exceeding the change threshold. The change threshold is defined by the `incrementalupdatethresholdpercent` attribute of the `databasestatistics` element in `database-config.xml`. Specify `false` to gather full database statistics. The description is shown on the **Execution History** tab of the **Database Statistics** info page. See "Configuring Database Statistics" on page 42.
`-user user`	Specifies the *`user`* to run the process.
`-verifydbschema`	Verifies that the data model matches the underlying physical database.
`-version`	Returns the running server version, the database schema version, and configuration version.

Table Import Command

```
table_import -help
table_import -password password [-server url] [-user user] { -D name=value |
             { [-batch] { -deleteexcluded | -updatedatabasestatistics | -integritycheckandload
             [-allreferencesallowed | -clearerror | -estimateorastats | -populateexclusion]
             -clearexclusion | -clearstaging | -outputerrors filename | -zonedataonly}
table_import -password password [-server url] [-user user] { -D name=value |
             {[-batch] { -deleteexcluded | -integritycheck [-allreferencesallowed | -clearerror |
             -populateexclusion | -updatedatabasestatistics ] -clearexclusion -clearstaging |
             -outputerrors filename }
```

The `table_import` command loads data from staging tables into PolicyCenter. Before you can use this command, use the system tools command to set the server run level to `MAINTENANCE`.

IMPORTANT PolicyCenter supports bulk data import only for loading zone data from staging tables. For more information, see "Zone Import Command" on page 169.

See also

- "System Tools Command" on page 162.
- For more complete information on importing zone data and database staging tables generally, see "Zone Import" on page 541 in the *Integration Guide*
- For information on the web service `TableImportAPI` that also loads data from staging tables, see "Table Import Tools" on page 548 in the *Integration Guide*.

Table Import Options

You can use any of the following options with the `table_import` command. You must always supply the `-password` option.

Option	Description
`-D name=value`	Sets a Java system property.
`-allreferencesallowed`	Allows references to existing rows in all source tables, including administrative tables such as users and groups.

Option	Description
-batch	Runs the command in a batch process. This flag only applies with the following: -integritycheck -integritycheckandload -populateexclusion -deleteexcluded -updatedatabasestatistics
-clearerror	Clears the error table.
-clearexclusion	Clears the exclusion table.
-clearstaging	Clear the staging tables.
-deleteexcluded	Deletes rows from staging tables based on contents of exclusion table.
-estimateorastats	This parameter applies only to Oracle databases. It updates database statistics on the source tables with estimated row and block counts for the source tables and indexes at the beginning of load (-integritycheckandload).
-help	Displays the command usage.
-integritycheck	Validates the contents of the staging tables. You can optionally specify: -allreferencesallowed -clearerror -populateexclusion
-integritycheckandload	Validates the contents of the staging tables and populate source tables. You can optionally specify one of the following flags: -allreferencesallowed -clearerror -estimateorastats -populateexclusion -zonedataonly
-messagesinks *sinks*, ...	Deprecated. This flag does not do anything.
-password *password*	Specifies the administrative password. You must specify a *password*.
-populateexclusion	Populate the exclusion table with rows to exclude.
-server *url*	Specifies the PolicyCenter host server url.
-updatedatabasestatistics	Updates the database statistics on the staging tables. If you also specify the -integritycheckandload option, this option calculates the estimated row and block counts for the source tables. When running against Oracle, this command updates indexes before populating.
-user *user*	Specifies the *user* to run the process.
-zonedataonly	Sets the import to load zone data only. Used with the -integritycheckandload flag.

Template Tools Command

```
template_tools -help
template_tools -password password [-server url] [-user user] [ -D name=value ]
               [ -working_dir name] { -convert_dir directory | -convert_file filename |
               -list_template | -import_dir objectsfile fieldsfile directory |
               -import_files objectsfile fieldsfile outfile | -validate_all | -validate_template id }
```

Template Tools Options

You can use any of the following options with the `template_tools` command. You must always supply the `-password` option.

Option	Description
-help	Prints a help message.
-convert_dir *directory*	Converts templates in the specified directory to the new format.
-convert_file *filename*	Converts the specified template to the new format.

Option	Description
`-import_dir objectsfile fieldsfile directory`	Imports context objects and form fields into all the templates in the specified *directory*.
`-import_files objectsfile fieldsfile outfile`	Imports context objects and form fields into all the templates in the specified template *outfile*.
`-list_templates`	Lists all of the templates available for validation.
`-password password`	Specifies the administrative password. You must specify a *password*.
`-server url`	Specifies the PolicyCenter host server url.
`-user user`	Specifies the *user* to run the process.
`-validate_all`	Validates all the templates.
`-validate_template id`	Validates a single template.
`-working_dir directory`	Specify a directory so that relative paths can be used for file arguments.

Usage Tools Command

```
usage_tools -help
usage_tools -password password [-server url] [-user user] [-D name=value | -count_exposures
            [-generate_csv | {-include_exposure_details | -include_monthly_details}] |
            -start_date startdate | -end_date enddate ]
```

Usage Tools Options

You can use any of the following options with the `usage_tools` command. You must always supply the `-password` option.

Option	Description
`-count_exposures`	Count the total number of exposures.
`-D name=value`	Sets a Java system property.
`-end_date enddate`	Counts the total number of exposures up to and including this date. You can only specify this option with the `-count_exposures` option.
`-help`	Displays the command usage.
`-include_exposure_details`	Outputs all the exposure details. You cannot use this with the `-include_monthly_details` option.
`-include_monthly_details`	Includes the exposures per month in the output. You cannot use this with the `-include_monthly_details` option.
`-password password`	Specifies the administrative password. You must specify a *password*.
`-start_date startdate`	Counts the total number of exposures from this date forward. You can only specify this option with the `-count_exposures` option.
`-server url`	Specifies the PolicyCenter host server url.
`-user user`	Specifies the *user* to run the process.

Workflow Tools Command

```
workflow_tools -help
workflow_tools -password password [-server url] [-user user] {[ -D name=value |
               -cancel id | -complete id | -restart id | -schedule_now id }
```

Workflow Tools Options

You can use any of the following options with the `workflow_tools` command. You must always supply the `-password` option.

`-cancel id`	Cancels a running workflow for the specified workflow *id*.
`-complete id`	Completes running workflow for the specified workflow *id*.
`-D name=value`	Sets a system property.
`-help`	Displays the command usage.
`-password password`	Specifies the administrative password. You must specify a *password*.
`-restart_all`	Restart all workflows in the Error state.
`-restart id`	Restarts the workflow with the specified *id* in the Error state.
`-schedule_now id`	Schedules workflow with the specified *id* immediately.
`-server url`	Specifies the PolicyCenter host server url.
`-user user`	Specifies the *user* to run the process.

You can also control workflows using `WorkflowAPI`. See “Workflow Web Services” on page 98 in the *Integration Guide*.

Zone Import Command

```
zone_import -help
zone_import -password password [-server url] [-user user] { -D name=value |
            {-import filename -country country [-clearstaging] [-charset charset]}
zone_import -password password [-server url] [-user user] { -D name=value |
            {-clearstaging [-country country]}
zone_import -password password [-server url] [-user user] { -D name=value |
            {-clearproduction [-country country]}
```

The `zone_import` command imports data in CSV format from specified files into database staging tables for zone data. You can import zone data for one country at a time only. The zone data files that you import must contain zone data for a single country only. You can load zone data for multiple countries by using the command multiple times with different, country-specific zone data files each time.

Guidewire expects that you import address zone data upon first installing the product and at infrequent intervals thereafter as you receive data updates.

See also

- For more complete information on importing zone data and database staging tables generally, see “Zone Import” on page 541 in the *Integration Guide*.
- For information on the web service `ZoneImportAPI` that also imports zone data, see “Introduction to Zone Import” on page 541 in the *Integration Guide*.

Importing a Zone Data File

Before you can use the `zone_import` command, set the PolicyCenter server run level to `MAINTENANCE`. After you use the command to import zone date, move the data from the staging tables to the production tables by using the `table_import` command.

To import a zone data file

1. Start the PolicyCenter server.

2. Set the PolicyCenter server run level to `MAINTENANCE`:

   ```
   system_tools -password password -maintenance
   ```

3. Clear the zone data staging tables. If you have multiple countries defined, you can include the `-country countryCode` option to clear staging zone data only for the country you will be loading:

   ```
   zone_import -password password -clearstaging [-country countrycode]
   ```

4. Load the zone data file into the staging tables:

   ```
   zone_import -password password -country countrycode -import filename
   ```

5. Clear existing zone data in production. This is useful if there is already zone data for the particular country that you are loading:

   ```
   zone_import -password password -country countrycode -clearproduction
   ```

6. Load zone data from the staging tables into production:

   ```
   table_import -password password -integritycheckandload -zonedataonly
   ```

7. Set the server run level back to `MULTIUSER`:

   ```
   system_tools -password password -multiuser
   ```

Zone Import Options

You can use any of the following options with the `zone_import` command. You must always supply the `-password` option.

Option	Description
`-charset charset`	Character set encoding of the zone data file. The default is UTF-8.
`-clearproduction`	Clears zone data from the production tables. Optionally, specify the `-country` parameter to clear data for only one country.
`-clearstaging`	Clears zone data from the staging tables. Optionally, specify the `-country` parameter to clear data for only one country.
`-country countrycode`	• If you specify the `-import` parameter, the country of the zone data in the import file. • If you specify the `-clearproduction` and/or `-clearstaging` parameters, the country of the zone data to clear from the tables. • To clear the production and staging tables for a country and import new zone data for that country, specify the `import`, `-clearproduction`, and `-clearstaging` parameters, in addition to this parameter.
`-D name=value`	Sets a Java system property.
`-help`	Displays the command usage.
`-import filename`	Imports zone data from the specified file. You must set a value for the `-country` parameter.
`-password password`	Specifies the administrative password. You must specify a password.
`-server url`	Specifies the URL of the PolicyCenter host server.
`-user user`	Specifies the user under whose authorization the zone import process runs.

Zone Data Files Supplied by Guidewire

PolicyCenter stores the `zone-config.xml` files in the `geodata` folder, each in its own individual country folder. For example, the `zone-config.xml` file that configures address-related information in Australia is in the following location in the Studio **Project** window:

configuration → **config** → **geodata** → **AU**

PolicyCenter provides a collection of zone data files for various localities with small sets of zone data that you can load for development and testing purposes. The zone data files are in the following location in the Studio **Project** window:

configuration → config → geodata

Guidewire provides the `US-Locations.txt` and similar files for testing purposes to support autofill and autocomplete when users enter addresses. This data is provided on an as-is basis regarding data content. For example, the provided zone data files are not complete and may not include recent changes.

Also, the formatting of individual data items in these files might not conform to your internal standards or the standards of third-party vendors that you use. For example, the names of streets and cities are formatted with mixed case letters but your standards may require all upper case letters.

The `US-Locations.txt` file contains information that does not conform to United States Postal Service (USPS) standards for bulk mailings. You can edit the `US-Locations.txt` file to conform to your particular address standards, and then import that version of the file.

Zone Data Files That You Create

You can create your own zone data files, in CSV format. The import tool uses configuration information from `zone-config.xml` files for specific countries to determine which data fields to import and what each field represents for each country. The `zone-config.xml` files for specific countries are located in folders for specific countries. Navigate in the Studio **Project** window to **configuration → config → geodata.**

The following example is the zone configuration file for zone data for France.

configuration → config → geodata → FR → `zone-config.xml`

Each `zone-config.xml` files must have one `Zones` element for a specific country. The `Zone` element defines the fields in zone data files for a country. For each field, the `fileColumn` attribute indicates the position of the field within lines of the files.

The following example XML code from a `zone-config.xml` file defines the fields to import from zone data files for the United States. The example code specifies for United States zone data files that the third field in the comma-separated values of each line of corresponds to a city.

```
<Zones countryCode="US">
  ...
  <Zone code="city" fileColumn="3" granularity="2">
  ...
```

See also

- For complete information about `zone-config.xml`, including a description of its XML elements and attributes, see “Configuring Zone Information” on page 122 in the *Globalization Guide*.

chapter 13

Free-text Batch Load Command

The free-text batch load command loads the Guidewire Solr Extension, a full-text search engine, with *index documents* for all policies in your PolicyCenter application database. Index documents are XML documents that contain a subset of the information from policies in PolicyCenter. The Guidewire Solr Extension indexes the documents after it receives them from the free-text batch load command.

> **Note:** The free-text batch load command runs on the host where the Guidewire Solr Extension resides. The command is located in the `/opt/gwsolr/pc/solr/policy_active/conf` directory, not the `PolicyCenter/admin/bin` directory.

This topic includes:

See also

- "Free-text Search Setup" on page 89 in the *Installation Guide*
- "Configuring the Free-text Batch Load Command" on page 350 in the *Configuration Guide*

When to Run the Free-text Batch Load Command

Generally, PolicyCenter updates the Guidewire Solr Extension whenever someone changes a policy and that change affects index documents stored there. In response, the Guidewire Solr Extension incrementally indexes the changed information. Occasionally, you need to load the Guidewire Solr Extension and have it build new indexes based on the newly loaded information.

IMPORTANT Users must not perform free-text searches while the free-text batch load command runs. Otherwise, the search results will be incomplete. Set the `EnableDisplayBasicSearchTab` script parameter to `false` to temporarily hide the free-text search user interface.

Run the free-text batch load command whenever any of the following occur:

- Policies are bulk loaded into the PolicyCenter application database.
- Indexes become corrupted in the Guidewire Solr Extension, as reported by the Guidewire Solr Extension web application.
- Changes are made to the metadata definitions of entities and relationships in the policy graph, and these changes affect index documents stored in the Guidewire Solr Extension.
- Changes are made to attribute definitions in `schema.xml`.
- Changes to the mapping (`policy-search-config.xml` or custom mappers) that affect already indexed periods.
- Your instance of PolicyCenter is upgraded to a later version, and the upgrade changes metadata definitions for index documents stored in the Guidewire Solr Extension.

Do not run the free-text batch load command if you configured free-text search for embedded operation. Whenever the Guidewire Solr Extension runs in embedded mode, use the **Free-text Search** page on the **Server Tools** tab.

See also

- "Free-text Search" on page 154

Prerequisites for Running the Free-text Batch Load Command

Before you run the free-text batch load command for the first time, you must modify the following files:

- `data-config.xml` – Specifies for the batch load command the location of the collated and compiled index documents for the Guidewire Solr Extension to load. It also specifies the fields the index documents contain.
- `batchload.sh/batchload.bat` – Specifies the `batchload-config-`*`databaseBrand`*`.xml` configuration file to use for your database brand.
- `batchload-config-`*`databaseBrand`*`.xml` – Specifies the SQL Select statement that extracts policies from the PolicyCenter application database. Specifies the URL for the Guidewire Solr Extension. Specifies a working directory, and specifies a sort binary.

Each time before you run the free-text batch load command, you must do all of the following if PolicyCenter is running:

- Suspend the `PCSolrMessageTransport` message destination from the **Event Messages** page on the **Administration** tab.

  Suspending the message destination prevents PolicyCenter from sending updated index documents to the Guidewire Solr Extension if users modify policies while the free-text batch load command runs.
- Set the `EnableDisplayBasicSearchTab` script parameter to `false` from the **Script Parameters** page on the **Administration** tab.

Setting the script parameter to `false` prevents users from accessing the **Basic** tab to perform free-text searches while the free-text batch load command runs.

Running the Free-text Batch Load Command

You run the free-text batch command on the host where the Guidewire Solr Extension resides. Run the command only if you configured free-text search for external operation.

To run the free-text batch load command

1. In PolicyCenter, do the following:
   a. Suspend the `PCSolrMessageTransport` message destination.
   b. Set the `EnableDisplayBasicSearchTab` script parameter to `false`.
2. Shut down and restart the Guidewire Solr Extension.
   Shutting down the Guidewire Solr Extension forces it to pick up any changes to `data-config.xml`.
3. Switch to the `/opt/gwsolr/pc/solr/policy_active/conf` directory.
4. Run the `batchload` command.
5. After the command finishes, examine the status response to verify that your load succeeded.
   A problem-free load gives the same positive number for Total Rows Fetched and Total Documents Processed.
6. In PolicyCenter, do the following:
   a. Resume the `PCSolrMessageTransport` message destination.
   b. Set the `EnableDisplayBasicSearchTab` script parameter to `true`.

Recovering from Errors

The free-text batch load command queries the PolicyCenter database for data and then locally processes the information intensively on disk. The command eventually produces an XML file with index documents ready for the Guidewire Solr Extension. In the last step, the command tells the Guidewire Solr Extension to load the index documents.

The batch load command can fail in the last step and end without loading any index documents. For example, an error in `data-config.xml` can cause the load to fail. In such cases, you do not need to run the entire batch load command again. Instead, you can invoke the Guidewire Solr Extension directly to complete its portion of the batch load process by using the following URL:

```
http://hostName:8983/pc-gwsolr/pc_policy_active/dataimport?command=full-import&entity=policy
```

Clean-up Tasks after Running the Free-text Batch Load Command

Each time after you run the free-text batch load command, you must do all of the following:

- Resume the `PCSolrMessageTransport` message destination from the **Event Messages** page on the **Administration** tab.
  Resuming the message destination lets PolicyCenter send updated policy data to the Guidewire Solr Extension for incremental indexing.
- Set the `EnableDisplayBasicSearchTab` script parameter to `true` from the **Script Parameters** page on the **Administration** tab.
  Setting the script parameter to `true` lets users access the **Basic** tab to perform free-text searches.
- Consider deleting files from the `loadDir` directory.

Free-text Batch Load Command and Native SQL

The free-text batch load command extracts all policy data from the PolicyCenter application database by using native SQL. The SQL Select statement that the batch load command uses is defined in configuration files for specific database brands. These configuration files are located on the host where the Guidewire Solr Extension resides, in the following directory:

```
opt/gwsolr/pc/solr/policy_active/conf
```

The configuration files that contain native SQL are:

- **For H2 databases** – `batchload-config-h2.xml`, suitable only for development
- **For Oracle databases** – `batchload-config-oracle.xml`, suitable for development or production
- **For SQL Server databases** – `batchload-config-sqlserver.xml`, suitable for development or production

See also

- "SQL Select Statement Configuration for the Free-text Batch Load Command" on page 351 in the *Configuration Guide*