Guidewire PolicyCenter™

System Administration Guide

Release 10.0.0

GUIDEWIRE
Adapt and succeed™

Product Name: Guidewire PolicyCenter
Product Release: 10.0.0
Document Name: System Administration Guide
Document Revision: 25-September-2018

Contents

Part 3
Server clustering administration

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.
Database Upgrade Guide	Describes the overall PolicyCenter upgrade process, and describes how to upgrade your PolicyCenter database 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.
Configuration Upgrade Guide	Describes the overall PolicyCenter upgrade process, and describes how to upgrade your PolicyCenter configuration 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. The *Configuration Upgrade Guide* is published with the Upgrade Tools and is available from the Guidewire Community.
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 for PolicyCenter. The intended readers are all IT staff and configuration engineers.
PCF Reference Guide	Describes PolicyCenter PCF widgets and attributes. The intended readers are configuration engineers.
Data Dictionary	Describes the PolicyCenter data model, including configuration extensions. The dictionary can be generated at any time to reflect the current PolicyCenter configuration. The intended readers are configuration engineers.
Security Dictionary	Describes all security permissions, roles, and the relationships among them. The dictionary can be generated at any time to reflect the current PolicyCenter configuration. The intended readers are 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 Guidewire Studio for PolicyCenter. 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.

Document	Purpose
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.
Java API Reference	Javadoc-style reference of PolicyCenter Java plugin interfaces, entity fields, and other utility classes. The intended readers are system architects and integration programmers.
Gosu Reference Guide	Describes the Gosu programming language. The intended readers are anyone who uses the Gosu language, including for rules and PCF configuration.
Gosu API Reference	Javadoc-style reference of PolicyCenter Gosu classes and properties. The reference can be generated at any time to reflect the current PolicyCenter configuration. The intended readers are configuration engineers, system architects, and integration programmers.
Glossary	Defines industry terminology and technical terms in Guidewire documentation. The intended readers are everyone who works with Guidewire applications.
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	Indicates a term that is being defined, added emphasis, and book titles. In monospace text, italics indicate a variable to be replaced.	A *destination* sends messages to an external system. Navigate to the PolicyCenter installation directory by running the following command: `cd installDir`

bold	Highlights important sections of code in examples.	`for (i=0, i<someArray.length(), i++) {`
		`  newArray[i] = someArray[i].getName()`
		`}`

narrow bold	The name of a user interface element, such as a button name, a menu item name, or a tab name.	Click **Submit**.
`monospace`	Code examples, computer output, class and method names, URLs, parameter names, string literals, and other objects that might appear in programming code.	The `getName` method of the `IDoStuff` API returns the name of the object.
`monospace italic`	Variable placeholder text within code examples, command examples, file paths, and URLs.	Run the `startServer` *`server_name`* command. Navigate to `http://`*`server_name`*`/index.html`.

Support

For assistance, visit the Guidewire Community.

Guidewire customers

`https://community.guidewire.com`

Guidewire partners

`https://partner.guidewire.com`

part 1

Application administration

chapter 1

Managing users

This topic discusses the default system users that Guidewire provides in the base PolicyCenter configuration.

PolicyCenter default system users

In the base configuration, PolicyCenter provides the following default system users:

- `defaultuser` - Default User
- `policychange_daemon` - PolicyChange Daemon
- `renewal_daemon` - Renewal Daemon
- `su` - Super User
- `sys` - System User

  Note: PolicyCenter default system users are separate and distinct from the users in the sample data that Guidewire provides for testing and development purposes.

Guidewire provides each of these default users for a specific purpose and to fulfill one or more specific roles. In most cases, it is not possible to delete a default user as internal code uses that user for a specific purpose. It is possible to edit various aspects of user information, for example, setting a new password for that user.

defaultowner

Default system user `defaultowner` has the following characteristics.

User	Default Owner
Username	`defaultowner`
Group	Default Root Group
Can delete	No

User `defaultowner` has first name Default and last name Owner. In the base configuration, PolicyCenter does not assign roles to this user.

If PolicyCenter cannot assign certain business objects to a specific user, PolicyCenter assigns that object to user `defaultowner`. PolicyCenter performs this assignment internally.

IMPORTANT Do not make direct assignments to user `defaultowner`.

Guidewire recommends, as a business practice, that someone in the organization periodically search for outstanding work assigned to user `defaultowner`. If the search finds one of these assignments, the searcher must reassign these items to a proper owner. Guidewire also recommends that the rule administrator investigate why PolicyCenter did not assign an item of that type and correct any errors in the rules.

policychange_daemon

Default system user `policychange_daemon` has the following characteristics.

User	PolicyChange Daemon
Username	`policychange_daemon`
Group	Default Root Group
Can delete	Yes

User `policychange_daemon` has first name PolicyChange and last name Daemon. In the base configuration, PolicyCenter assigns the `superuser` role to the `policychange_daemon` user. The `superuser` role has all permissions. Thus, user `policychange_daemon` has unrestricted access to the entire PolicyCenter application.

PolicyCenter defines user `policychange_daemon` as the default user for automatic policy changes started through the `PolicyChangeAPI`. The `superuser` role assigned to this user provides overriding authority to auto-approve all underwriting issues that occur in automated policy changes. Guidewire recommends that you review the roles and permissions associated with this user in a production environment.

Deleting user policychange_daemon

Although it is possible to delete user `policychange_daemon`, the code in `PolicyChangeAPI.gs` explicitly names this user. If you delete this user, you must update the code in `PolicyChangeAPI` with the name of a user with the equivalent permissions.

renewal_daemon

Default system user `renewal_daemon` has the following characteristics.

User	Renewal Daemon
Username	`renewal_daemon`
Group	Default Root Group
Can delete	Yes

User `renewal_daemon` has first name Renewal and last name Daemon. In the base configuration, PolicyCenter assigns the `superuser` role to the `renewal_daemon` user. The `superuser` role has all permissions. Thus, user `renewal_daemon` has unrestricted access to the entire PolicyCenter application.

PolicyCenter defines user `renewal_daemon` as the default user for automated renewal policy transactions started through `PolicyRenewalStart` batch processing. The `superuser` role assigned to this user provides overriding authority to auto-approve all underwriting issues that occur in automated policy renewals. Guidewire recommends that you review the roles and permissions associated with this user in a production environment.

Deleting user renewal_daemon

Although it is possible to delete user `renewal_daemon`, the code in `PolicyRenewalPlugin.gs` (called by `PolicyRenewalStart`) explicitly names this user. If you delete this user, you must update the code in `PolicyRenewalPlugin` with the name of a user with the equivalent permissions.

su

Default system user `su` has the following characteristics.

User	Super User
Username	su
Group	Default Root Group
Can delete	No

User `su` has first name Super and last name User. In the base configuration, Guidewire assigns the Superuser role, with all permissions, to this user. Thus, user `su` has unrestricted access to the entire PolicyCenter application. It is not possible to add or remove permissions from this user.

IMPORTANT Guidewire strongly recommends that you change the Super User password from its default value.

User su and the command line tools

To run the PolicyCenter command prompt tools, you must supply a username and password. If you do not supply the `-user` parameter, the command defaults to user `su` and you must supply that user's password.

See also

- "Change the unrestricted user" on page 24
- "Command prompt tools" on page 421
- *Installation Guide*

sys

Default system user `sys` has the following characteristics.

User	System User
Username	sys
Group	Default Root Group
Can delete	No

User `sys` has first name System and last name User. In the base configuration, PolicyCenter does not assign roles to this user.

PolicyCenter requires user `sys` to exist. PolicyCenter uses this user to perform automated work such as running batch processing, messaging polling, and server startup. 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.

WARNING Do not rename or delete the `sys` user. Deleting or renaming this user disables PolicyCenter.

Temporary system users

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

PolicyCenter gives a designation of `Temporary system user` to an unauthenticated user session. PolicyCenter creates such sessions for login. By design, PolicyCenter does not associate a user with the login screen. The `system_tools -sessioninfo` command does not filter out this user. The Server Tools **Management Beans** screen does filter out this user.

Change the unrestricted user

About this task

By default, PolicyCenter provides user `su` (Super User) as the user with unrestricted access to the entire PolicyCenter application. It is possible to change the unrestricted user.

Procedure

1. In the Studio **Project** window, expand **configuration→config**:
2. Open file `config.xml` for editing.
3. Set configuration parameter `UnrestrictedUserName` to the name of the unrestricted user:

```
<param name="UnrestrictedUserName" value="userName"/>
```

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"/>
```

chapter 2

Logging in Guidewire PolicyCenter

Guidewire PolicyCenter uses Apache `log4j-2` libraries, in conjunction with the `slf4j` API and internal Guidewire libraries, to provide logging in the PolicyCenter application.

Overview of application logging

Guidewire uses the `slf4j` API, in conjunction with Apache `log4j-2` libraries and internal Guidewire libraries, to provide logging in the PolicyCenter application. An XML-formatted logging configuration file controls the behavior of the logging activity in the application.

For more information on `slf4j`, refer to the following SLF4J documentation:

```
http://slf4j.org/index.html
```

For more information on Apache `log4j-2`, refer to the following Apache documentation:

```
https://logging.apache.org/log4j/2.x/
```

See also

- “Logging level reference” on page 25
- “Working with logging categories” on page 31
- “The logging definition file” on page 26

Logging level reference

The logging level determines how much information PolicyCenter records in the log. The following list describes the different levels of information that PolicyCenter records in the log, in the increasing order of the severity of the information.

Level	Description
`TRACE`	Messages about processes that are about to start or that completed. These types of messages 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”.

Level	Description
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.” • “User X 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 PolicyCenter used the default value.” • “A plugin call took over 90 seconds.”
ERROR	Messages that indicate a definite problem. Typical messages might include: • “A remote system refused a connection to a plugin call.” • “PolicyCenter can not complete operation XYZ even with a default.”

The logging definition file

File `log4j2.xml` specifies the logging options for the PolicyCenter server. You access this file from the following location in Guidewire Studio:

configuration→config→logging

The `log4j2` file uses the XML format specified by Apache log4j-2. The entries in `log4j2.xml` control what to log and to which file to write the logging information.

This file contains a single top-level `<Configuration>` element with multiple subelements.

```
<Configuration status="...">
  <Properties .../>
  <Appenders .../>
  <Loggers .../>
</configuration>
```

Often, while configuring Apache log4j-2, it is necessary to view the generated status events. Adding the `status` attribute to the `<Configuration>` element provides a way to view status events at the specified level.

The logging elements define the following logging components:

`<Properties>`	A *property* defines a specific global value in the logging file. For example: • The base directory to which PolicyCenter writes the generated log files. • The default formatting pattern to use for each log file name.
`<Appenders>`	An *appender* defines a specific target location to which to write logging information. The appender definition includes such information as the log file name and the text formatting to use in the file. Examples of appenders include: • The `Console` appender, which defines the application console log. • The `RuleEngineLog`, which defines a log file that records logging messages generated by the Rules Engine.
`<Loggers>`	A *logger* defines configuration parameters associated with a specific appender. For example, this can be the logging level for a specific logging category associated with an appender.

For more information on Apache `log4j-2` logging components, refer to the following Apache documentation:

```
http://logging.apache.org/log4j/2.x/manual/configuration.html
```

Logging property definitions

Each `Property` element in file `log4j2` defines a specific global variable that PolicyCenter uses in generating the application logs. The `Property` element uses the following syntax to define a property.

```
<Configuration>
  <Properties>
    <Property name="propertyName">propertyValue</Property>
</Properties>
...
</Configuration>
```

In the base configuration, PolicyCenter defines the following logging-related properties.

`guidewire.logDirectory`	Defines the default directory to which PolicyCenter writes all logging files.
`file.defaultPattern`	Defines the default formatting to use in building the names of the log file that PolicyCenter generates on a daily basis.

To use a property, insert the property name as a variable. For example, to use the default directory path as a variable, use the following syntax.

```
${guidewire.logDirectory}
```

See also

- "The default log file directory property" on page 27
- "The default text formatting property" on page 28

The default log file directory property

In the base configuration, file `log4j2.xml` defines the default logging directory as `/tmp/gwlogs/PolicyCenter/logs` by using the following `Property` element.

```
<Property name="guidewire.logDirectory">/tmp/gwlogs/PolicyCenter/logs</Property>
```

PolicyCenter uses these directory `Property` values in configuring the appenders that generate the log files. For example, the following definition for a `RollingFile` appender uses the default logging directory path.

```
<RollingFile name="DailyFileLog" fileName="${guidewire.logDirectory}/pclog.log"
      filePattern="${guidewire.logDirectory}/pclog.log%d{.yyyy-MM-dd}">
  ...
</RollingFile>
```

Notice how this code defines the default directory path as a variable.

```
${guidewire.logDirectory}
```

It is possible to change the default directory location by modifying the directory path.

File path definition

Express all file locations in file `log4j2` by using an absolute path. Regardless of the 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.

Note: If you specify a non-existent logging directory path, PolicyCenter writes logging information to the application console only.

The default text formatting property

File `log4j2.xml` defines the format to use in formatting logging messages by using the following `Property` element.

```
<Property name="file.defaultPattern">%-10.10X{server} %-8.24X{userID}
      %d{yyyy-MM-dd HH:mm:ss,SSS} %p %notEmpty{&lt;%marker&gt; }%m%n
</Property>
```

Notice that conversion patterns use control characters, similar to the C language `printf` function, to specify the output format for the message. For example, `%-10.10X{server}` has the following meaning:

- `%-10.10` – Pad the text with spaces to the right if the output is shorter than 10 characters. Truncate the output if the output is longer than 10 characters, starting from the left-hand side of the text.
- `%X{server}` – Add the server name to the output text, replacing the variable `{server}` with the name of the actual server.

For more information on conversion patterns, refer to the following Apache documentation:

```
https://logging.apache.org/log4j/1.2/apidocs/org/apache/log4j/PatternLayout.html
```

See also

- "Conversion character reference" on page 38
- "Format modifier reference" on page 39

Logging appender definitions

Periodically, PolicyCenter creates a new, blank, log file and begins writing the current logging information to that file. It retains the log file from the previous time period as a backup, renaming the file to make the file name distinctive. This mechanism is known as file rollover.

In the base configuration, file `log4j2` contains a number of `<RollingFile>` appender elements that define different log files. For example, file `log4j2` configures one log to record rule activity and a different log to record all API calls. However, in the base configuration, Guidewire comments out most of the `RollingFile` appenders. To make a log appender active, you must remove the comment marks that surround the appender.

Each `<RollingFile>` appender element contains the following information:

- The location to which to write the log
- The pattern to use to create the log file name
- The pattern to use to create the backup log file name
- The pattern to use to create formatted content in the log file
- The time interval at which to roll over the current contents of the log file to a backup log file

See also

- "Daily File Log Definition" on page 29
- "How to format a log message" on page 37

Root logging appender definitions

File `log4j2.xml` contains two root logging appender definitions.

`Console`	Defines how PolicyCenter sends logging messages to the application console.

`DailyFileLog`	Defines the main, rollover log for the PolicyCenter application.

File `log4j2.xml` defines the `Console` and `DailyFileLog` appenders as root logging categories by using the `<Root>` subelement on the `<Loggers>` element.

```
<Configuration>
  <Loggers>
    <Root level="info">
      <AppenderRef ref="Console"/>
      <AppenderRef ref="DailyFileLog"/>
    </Root>
    ...
  </Loggers>
</Configuration>
```

Notice the following:
- The `level` attribute sets the logging level to `info` for all the listed appenders and those appenders that inherit from listed appenders.
- The `ref` attribute specifies the name of the associated appender.

See also

- “Default logging categories” on page 32
- “Console Appender Definition” on page 29
- “Daily File Log Definition” on page 29

Console Appender Definition

File `log4j2` defines a `Console` appender, which is the root application console log definition. In the base configuration, the `Console` appender definition is similar to the following:

```
<Configuration>
  <Appenders>
    <Console name="Console" follow="true">
      <PatternLayout pattern="%-10.10X{server} %-8.24X{userID}
            %d{yyyy-MM-dd HH:mm:ss,SSS} %p %notEmpty{&lt;%marker&gt; }%m%n" charset="UTF-8"/>
    </Console>
  </Appenders>
</Configuration>
```

Notice that the `Console` appender uses its own `PatternLayout` definition to format the output text. It does not use the default pattern that property `file.defaultPattern` defines.

See also

- “The default text formatting property” on page 28
- “Logging appender definitions” on page 28
- “Daily File Log Definition” on page 29

Daily File Log Definition

File `log4j2.xml` defines a default daily file log (`DailyFileLog`), which is the root logging file for Guidewire PolicyCenter. In the base configuration, the `DailyLogFile` appender definition is similar to the following:

```
<Configuration>
  <Appenders>
    <RollingFile name="DailyFileLog" fileName="${guidewire.logDirectory}/pclog.log"
          filePattern="${guidewire.logDirectory}/pclog.log%d{.yyyy-MM-dd}">
      <PatternLayout pattern="${file.defaultPattern}" charset="UTF-8"/>
      <TimeBasedTriggeringPolicy/>
    </RollingFile>
  </Appenders>
</Configuration>
```

Notice the following:

- Attribute `fileName` on `<RollingFile>` defines the daily log file name. The attribute includes the full directory path that global property `guidewire.logDirectory` defines.
- Attribute `filePattern` on `<RollingFile>` defines the log file name of the archive backup file. The filename definition contains a variable that PolicyCenter updates every time that it creates a new backup file.
- Attribute `pattern` on `<PatternLayout>` defines the formatting conventions to use in writing information to the log file. In this example, it uses the default formatting defined in property `file.defaultPattern`.
- Element `<TimeBasedTriggeringPolicy>` defines the frequency at which PolicyCenter performs the file rollover process. If not set, as in this case, it defaults to rolling over the log file at midnight each night.

Refer to the following Apache documentation for details of how to use `RollingFile` appenders and the `<TimeBasedTriggeringPolicy>` element.

```
https://logging.apache.org/log4j/2.x/manual/appenders.html
```

See also

- “Console Appender Definition” on page 29
- “Logging appender definitions” on page 28

Logger definitions

You use a `<Logger>` element in file `log4j-2.xml` to define configuration parameters for an associated `<RollingFile>` appender. Using a `<Logger>` element, you can set the logging level for a specific logging category associated with an appender, for example.

The following code samples illustrate this concept. In the first code sample, the `<RollingFile>` appender defines a file to store log messages generated by the Rule Engine as it executes Gosu in the application server. PolicyCenter then writes all messages sent to any of the `RuleEngine.*` logging categories to this log file.

```
<RollingFile name="RuleEngineLog" fileName="${guidewire.logDirectory}/ruleengine.log"
      filePattern="${guidewire.logDirectory}/ruleengine.log%d{.yyyy-MM-dd}">
  <PatternLayout pattern="${file.defaultPattern}" charset="UTF-8"/>
  <TimeBasedTriggeringPolicy/>
</RollingFile>
```

The following associated `<Logger>` element sets several configuration parameters related to the `<RuleEngineLog>` appender.

```
<Logger name="RuleEngine" additivity="false" level="info">
  <AppenderRef ref="RuleEngineLog"/>
</Logger>
```

Notice the following:

- The `name` attribute defines the name of this `<Logger>` element.
- The `additivity` attribute value of `false` disables logging additivity, meaning that PolicyCenter does not propagate the logging message to any parents of the logging category.
- The `level` attribute sets the logging level to `info` for all appenders that inherit from the `RuleEngine` appender. For example, a logging category of `<RuleEngineLog.Gosu>` inherits the `info` log level from the parent `<RuleEngineLog>` appender, unless you specifically set the logging level elsewhere.
- The `ref` attribute specifies the name of the associated appender.

Refer to the following Apache documentation for more information.

```
https://logging.apache.org/log4j/2.x/manual/appenders.html
```

Working with logging categories

File `log4j2.xml` contains examples of how to set up logging for various logging categories. In the base configuration, Guidewire comments out most of these examples.

It is possible for both internal PolicyCenter code or custom integration code to define additional logging categories that do not show in the logging configuration file. It is also possible for third-party components, such as Apache, to provide their own logging categories. Each Guidewire application has its own unique categories as well.

In some cases, PolicyCenter does not use a particular plugin interface. In those cases, the logging category exists in file `log4j2.xml`, but, PolicyCenter does not use the category for logging.

Defining a logging category

You define a logging category in `log4j2.xml` file by using a `<Logger>` element, with each `<Logger>` element containing, at the minimum, the following items:

- The name of the logging category
- The log level for that logging category.
- A pointer to the associated `<Appender>` element that defines such items as the location of the log file and how to format the log message.

You use a `<Logger>` element to specify the name of the logging category and its log level. Typically, you use a `<RollingFile>` element to specify the output file name and location, and similar types of information. A rolling file is one that archives the current information at a specified time interval and then creates a new file to store the information going forward.

You need to specify both the `<Logger>` element and a `<RollingFile>` element in working with a logging category.

Working with the <Logger> element

The `<Logger>` element sets the name of the logger (the logging category), the log level for this logging category, and it points to the actual log file. The following example code is for the Database category.

```
<Logger name="Server.Database" additivity="false" level="debug">
  <AppenderRef ref="DatabaseLog"/>
</Logger>
```

Notice the following:

- The log category is `Server.Database`.
- The log level is `debug`.
- The log file name is `DatabaseLog`.

Working with the <RollingFile> element

A `<RollingFile>` element specifies the file name and file formatting of a log file. The following example code is for the `DatabaseLog` file.

```
<RollingFile name="DatabaseLog" fileName="${guidewire.logDirectory}/database.log"
      filePattern="${guidewire.logDirectory}/database.log%d{.yyyy-MM-dd}">
  <PatternLayout pattern="${file.defaultPattern}" charset="UTF-8"/>
  <TimeBasedTriggeringPolicy/>
</RollingFile>
```

Notice the following:

- The `name` attribute on the `<RollingFile>` element (`DatabaseLog`) matches the ref attribute on the associated `<Logger>` element.
- The `filename` attribute specifies the file path and file name of the database log and uses a variable for the file path.
- The `filePattern` attribute specifies the pattern to use in creating the log file name and again uses a variable for the file path.
- The `<PatternLayout>` element specifies how to format the text in the log file, using a variable to point to a default pattern.

Default logging categories

To ensure that all logging categories have a default logging level, file `log4j2.xml` defines logging levels for the main default loggers, `Console` and `DailyLogFile`, using the following `<Root>` element definition.

```
<Configuration>
  ...
  <Loggers>
    <Root level="info">
      <AppenderRef ref="Console"/>
      <AppenderRef ref="DailyLogFile" />
      ...
</Configuration>
```

Note: Every logging configuration must define at least one root logger. A root logger has no `name` or `additivity` attribute. It does have a `level` attribute.

Child loggers inherit their logging level from their parent logger, unless otherwise changed. In this way, you can achieve a fine-grained set of logging levels by logging category. The following code sample illustrates this concept.

```
<Logger name="Server.Archiving" level="info">
  <AppenderRef ref="ArchiveLog"/>
</Logger>
<Logger name="Server.Archiving.Success" level="info">
  <AppenderRef ref="ArchiveLog"/>
</Logger>
<Logger name="Server.Archiving.Graph" level="debug">
  <AppenderRef ref="ArchiveLog"/>
</Logger>
<Logger name="Server.Archiving.Graph" level="warn">
  <AppenderRef ref="Console"/>
</Logger>
<Logger name="Server.Archiving.DocumentUpgrade" level="trace">
  <AppenderRef ref="ArchiveLog"/>
</Logger>
```

In the example, `Server.Archiving` is the parent logger, which the logger sets to the `info` logging level. Child logger `Server.Archiving.Sucess` explicitly sets this same logging level.

However, notice that there are two loggers for `Server.Archiving.Graph`:

- One logger sets the logging level to `debug` for logging messages written to the archive log (`ArchiveLog`).
- The other logger sets the logging level to `warn` for logging messages written to the application console (`Console`).

The last logger, `Server.Archiving.DocumentUpgrade`, sets the logging level to `trace` and writes its logging messages to the archive log.

See also

- "Root logging appender definitions" on page 28
- For a information on logging levels and inheritance, refer to the Apache `log4j2` documentation:

```
https://logging.apache.org/log4j/2.x/
```

How to view a list of logging categories

You can use any of the following means to view a list of the currently enabled logging categories.

View	See…
Server Tools **Set Log Level** screen	"View logging categories using server tools" on page 33
`system_tools` command option	"View logging categories using system tools" on page 33
`SystemToolsAPI` web service	"View logging categories using web services" on page 34

See also

- "The Set Log Level screen" on page 361

View logging categories using server tools

Procedure

1. Log into PolicyCenter as a user with administrative privileges.
2. Navigate to the Server Tools **Set Log Level** screen.
3. Expand the **Logger** drop-down.

   In the drop-down list, you can view the PolicyCenter standard logging categories, logging categories for internal Guidewire code, and logging categories for third-party software.

See also

- *System Administration Guide*

View logging categories using system tools

About this task

The `system_tools` command lists logging categories defined by PolicyCenter only. It does not list third-party logging categories.

Procedure

1. Ensure that the PolicyCenter server is running.
2. Navigate to the following directory:

   ```
   admin/bin
   ```

3. Enter the following command:

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

   You must supply the username (*`user`*) and password (*`password`*) for a user with administrative privileges on the PolicyCenter server.

Result

PolicyCenter prints the PolicyCenter logging categories to the command prompt.

See also

- *System Administration Guide*

View logging categories using web services

About this task

The `SystemToolsAPI` method lists logging categories defined by PolicyCenter only. It does not list third-party logging categories.

Procedure

1. Ensure that the PolicyCenter server is running.
2. Call the following method on the `SystemToolsAPI` web service:

```
SystemToolsAPI.getLoggingCategories()
```

See also

- *Integration Guide*

Logging category reference

The following list describes some of the major Guidewire logging categories that are available in PolicyCenter. The list is not exhaustive.

Logging category	Description
`Api.*`	Base logging category for calls for all SOAP APIs.
`Application.*`	Base logging category for internal Guidewire application logging.
`Application.Addressbook`	Logging for Guidewire platform code that interacts with ContactManager. Guidewire ContactManager does not use this category.
`Assignment.*`	Base logging category for the assignment subsystem.
`Availability`	Logging of the determination of availability of an item in PolicyCenter. PolicyCenter bases the availability criteria for each item on the item type. For example, suppose that one criterion is the effective date of an item. If the effective date is not prior to or equal to today's date, the item is not available.
`BillingIntegration`	Logging of integration between PolicyCenter and BillingCenter.
`BizRules`	Base logging category for business rule activity and events.
`Configuration.*`	Base logging category for configuration problems in such areas as in security configuration, PCF configuration, locale configuration and so forth.
`Datagen`	Logging of internal Guidewire code used for testing.
`Geodata.*`	Base logging category for the Geodata daemon.
`Globalization.*`	Base logging category for the globalization subsystem.
`Import`	Logging for import of XML data into PolicyCenter.
`Integration.*`	Base logging category for general integration issues.
`LDAP`	Logging of issues related to the LDAP subsystem.
`Messaging.*`	Base logging category for the messaging system. PolicyCenter writes logging information in this category to a separate `messaging.log` file.
`OSGi.*`	Base logging category for calls to OSGi plugins.
`PerfAction.*`	Base logging category for issues related to performance.

Logging category	Description
`PerfAnalyzer`	Logging of issues related to the performance analyzer.
`Plugin.*`	Base logging category for all calls into any plugin. For child categories of `Plugin`, use the plugin name (`*PluginName*`), for example: `Plugin.*PluginName*` PolicyCenter writes logging information in this category to a separate `plugins.log` file.
`Profiler`	Base logging category for the Guidewire Profiler. See "The Guidewire Profiler screens" on page 407the *System Administration Guide* for more information on the Guidewire Profiler.
`PXLOGGER`	Logging for the internal Guidewire test platform.
`REST`	Base logging category for REST API calls.
`RuleEngine`	Do not use. Use `RuleExecution` instead.
`RuleExecution.*`	Base logging category for PolicyCenter rule execution. Only execution actions of rules generate logging events in this category. This category does not contain any information from the rules engine itself. PolicyCenter writes logging information in this category to a separate `ruleexecution.log` file.
`RuleExecutionUI`	Logging of rule execution activity in the PolicyCenter interface.
`Rules`	Do not use. Use the `RuleExecution` logging category instead.
`Security`	Internal Guidewire base logging category for security logging.
`Server.*`	Internal Guidewire base logging category for server and platform logging.
`Studio`	Logging of Guidewire Studio activity. This category applies to non-Gosu-related activities in Guidewire Studio only. However, if you execute Gosu code within the Studio Gosu Scratchpad, Studio executes the Gosu code on the server. Thus, it is possible to trigger Rule Engine logging on the server as well. PolicyCenter writes logging information in this category to a separate `studio.log` file.
`Test.*`	Internal Guidewire base logging category for the test subsystem.
`User`	Logging of each user's log in and log out of PolicyCenter.
`UserInterface`	Internal Guidewire base logging category for user interface logging.
`Workqueue`	Base logging category for work queue functionality. For more information on work queues, see "Administering PolicyCenter processes" on page 113the *System Administration Guide*.

See also

- "Working with logging categories" on page 31

Logging and Guidewire PolicyCenter

Guidewire PolicyCenter provides a number of ways to configure logging-related parameters.

This topic includes:

- "The log files directory and the View Logs screen" on page 36
- "Logging and server environments" on page 36
- "Configure logging in a multiple instance environment" on page 36

The log files directory and the View Logs screen

PolicyCenter includes a log viewer on the Server Tools **View Logs** screen, accessible to administrators. The `guidewire.logDirectory` property in file `log4j2.xml` specifies the default location of log files for the PolicyCenter log viewer.

In the base configuration, Guidewire sets this property to the following value:

```
guidewire.logDirectory = /tmp/gwlogs/PolicyCenter/logs/
```

Guidewire recommends that you set the log file locations for the individual logging appenders to the same directory as that defined in property `guidewire.logDirectory`. Using the same directory ensures that these log files are visible from the **View Logs** screen.

See also

- "The View Logs screen" on page 362

Logging and server environments

It is possible specify multiple `log4j2.xml` files that specify logging configuration for specific server environments. For example, it is possible to specify a logging configuration file for a test environment and a different configuration file for production environments:

- If the `env` attribute for a server is non-existent (`null`), PolicyCenter reads the logging configuration from the default `log4j2.xml` file.
- If the `env` attribute is non-null PolicyCenter tries to obtain the logging configuration from an *`env`*`-log4j2.xml` file that exists in the same directory as the default `log4j2.xml` file.

For example, if you have an server environment called `test`, PolicyCenter looks for logging configuration in file `test-log4j2.xml`. If this file does not exist, then PolicyCenter reads the logging configuration from the default `log4j2.xml` file.

See also

- "PolicyCenter server configuration" on page 59
- "Configure logging in a multiple instance environment" on page 36

Configure logging in a multiple instance environment

About this task

In file `log4j2.xml`, it is possible to use variables to specify log file names and locations. Using variables is particularly useful if there are multiple PolicyCenter instances on the same physical server. You can use a common logging definition file and generate log files for each separate PolicyCenter instance. You do this by defining a `serverid` system property in `config.xml` and using the system property in file `log4j2.xml`.

Procedure

1. In file `config.xml`, add an entry to the `<registry>` element that defines a `serverid` system property, for example:

   ```
   <registry roles="batch, workqueue, scheduler, messaging, startable, ui" >
     <systemproperty default="" name="serverid" value="myserverid" />
   </registry>
   ```

   Notice that this code defines a system property, `myserverid`.
2. Open file `log4j2` for editing.
3. If it does not already exist, use a `Property` element to set the default logging directory, for example:

```
<Property name="guidewire.logDirectory">/tmp/gwlogs/PolicyCenter/logs</Property>
```

You must use an absolute path to a directory that already exists. You must also use forward slashes as the path separator.

4. Update the `DailyFileLog` appender and set the `fileName` and `filePattern` attributes by using the system property that you defined in step 1. Use `serverid.noroles` to suppress the names of the roles associated with each server. Preface the name of the system property with `sys:`, for example:

```
  <RollingFile name="DailyFileLog" fileName="${guidewire.logDirectory}/${sys:myserverid.noroles}.log"
        filePattern="${guidewire.logDirectory}/${sys:myserverid.noroles}.log%d{.yyyy-MM-dd}">
  <PatternLayout pattern="${file.defaultPattern}" charset="UTF-8"/>
  <TimeBasedTriggeringPolicy/>
</RollingFile>
```

5. Rebuild and redeploy PolicyCenter to each cluster member.
6. Start each server by using the system property that sets the environment and server ID values. For example, if using development Jetty servers, use the following commands:

```
gwb runServer -Dgw.passthrough.myserverid=test#batch,messaging,ui
gwb runServer -Dgw.passthrough.myserverid=prod#batch, workqueue,scheduler,messaging,ui
```

Notice that you must use the Java system property `passthrough` syntax in starting the application server.

Result

As each servers starts, the server writes a log file to the common log file directory specified by the value of `guidewire.logDirectory` property. Each log file includes the value of `serverid` in the log file name. In this example, after starting the two defined servers, the `/tmp/gwlogs/PolicyCenter/logs` directory contains two log files:

- `test.log`
- `prod.log`

See also

- "Reloading the logging configuration" on page 40
- "Understanding the configuration <registry> element" on page 60
- "JVM options and server properties" on page 64
- *Installation Guide*

How to format a log message

Every `<RollingFile>` element in file `log4j2` contains a `<PatternLayout>` element that defines how to format the text of the logging information in each appender logging file. Apache utility class `PatternLayout`, a standard part of the Apache `log4j-2` distribution, provides the means of handling string patterns.

In the base configuration, each appender uses the default `file.defaultPattern` property to format the logging messages. The default pattern looks similar to the following code.

```
%-10.10X{server} %-8.24X{userID} %d{yyyy-MM-dd HH:mm:ss,SSS} %p %notEmpty{&lt;%marker&gt; }%m%n
```

You can, of course, modify the default pattern to meet your business needs. If you modify the default pattern, you change the text formatting in all log files globally, if not modified elsewhere. To modify the text formatting for an individual log category, modify the `<PatternLayout>` element associated with the appender for that logging category.

For more information on conversion patterns, refer to the following Apache documentation:

```
https://logging.apache.org/log4j/1.2/apidocs/org/apache/log4j/PatternLayout.html
```

See also

- "The default text formatting property" on page 28
- "Conversion character reference" on page 38
- "Format modifier reference" on page 39

Conversion character reference

Use the conversion characters listed in the following table to specify the formatting of logging category conversion patterns in file `log4j-2.xml`. For more information on conversion patterns, refer to the following Apache documentation:

```
http://logging.apache.org/log4j/2.x/manual/layouts.html
```

Character	Description
`%%`	Writes the percent sign to output.
`%c`	Name of the logging category (the logger name).
`%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.
`%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}` • ... 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`	New line character of the operating system. This is preferable to entering `\n` or `\r\n` as it works across platforms.
`%notEmpty`	Outputs the result of evaluating the pattern if and only if all variables in the pattern are not empty. In the context of logging markers, this is: `%notEmpty{&lt; %marker &gt;}`
`%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.

Character	Description
`%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` • `traceabilityID` 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.

See also

- "How to format a log message" on page 37
- "Format modifier reference" on page 39

Format modifier reference

File `log4j2.xml` defines how Guidewire PolicyCenter handles application logging. In the logging properties file, you can specify the format of information in log messages by using a conversion pattern for the characters. In using the conversion characters to define an output format, you can also add a format specification between the percent sign and the letter in the conversion pattern. The following table describes conversion patterns available with Apache `log4j2` libraries. For more information on conversion patterns, refer to the following Apache documentation:

Pattern	Description
`%`*`N`*	Specifies a minimum width of *N* for the output. *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. *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. *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.

Pattern	Description
`%M.N`	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`	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.

See also

- “Conversion character reference” on page 38

How to make dynamic changes to the logging configuration

It is possible, under certain circumstances, to make dynamic changes to the PolicyCenter logging configuration without having to redeploy PolicyCenter. Changing the database logging level dynamically does not make any change to existing database connections. The new logging level only takes effect for new database connections.

For example, if the database log level is set to debug, PolicyCenter logs all SQL statements. However, if you set the debug logging level dynamically, PolicyCenter only logs SQL statements for new connections created within the connection pool. For an existing connection, dynamically changing the logging level to debug has no affect.

To reset the logging configuration dynamically, do one of the following.

Change type	Persistence
Reset the log level from PolicyCenter, or by using system tools or a web service.	Temporary, ends at server restart
Update file `log4j.xml` and reload the file by using system tools or a web service.	Continues after server restart

You must use one of these methods to propagate logging configuration changes dynamically. Otherwise, to change a logging level, rebuild your PolicyCenter WAR or EAR and deploy it to the application server.

See also

- “Reloading the logging configuration” on page 40
- “Temporarily changing a logging level” on page 41
- *Installation Guide*

Reloading the logging configuration

It is possible to make an immediate change in the configuration of PolicyCenter logging without having to first redeploy the PolicyCenter application server. You can do this in several different ways:

- Use a `system_tools` command option
- Use a `SystemToolsAPI` web service method

Either approach reloads the current logging configuration from the `log4j2.xml` file.

See also

- “Reload the logging configuration by using system tools” on page 40
- “Reload the logging configuration by using web services” on page 41

Reload the logging configuration by using system tools

Procedure

1. Update file `log4j2.xml` with your logging configuration changes.
2. Ensure that the PolicyCenter server is running.

3. Open a command prompt in the following location:

```
admin/bin
```

4. Enter the following at the command prompt:

```
system_tools -user user -password password -reloadloggingconfig
```

You must supply the username (*user*) and password (*password*) for a user with administrative privileges.

See also

- *System Administration Guide*

Reload the logging configuration by using web services

Procedure

1. Update file `log4j2.xml` with your logging configuration changes.
2. Ensure that the PolicyCenter server is running.
3. Call the following method on the `SystemToolsAPI` web service:

```
SystemToolsAPI.reloadLoggingConfig()
```

See also

- *Integration Guide*

Temporarily changing a logging level

You can temporarily change the logging level for a logger by using one of the following options:

Option	More information
The `system_tools` command option	"system_tools command" on page 429
The `SystemToolsAPI` web service	*Integration Guide*
The Server Tools **Set Log Level** screen	"The Set Log Level screen" on page 361

Set the log level using system tools

Procedure

1. Ensure that the PolicyCenter application server is running.
2. Open a command prompt in the following location:

```
admin/bin
```

3. Enter the following at the command prompt:

```
system_tools -updatelogginglevel logger level
```

The PolicyCenter administration command prompt tools all require that you enter the password of an administrative user for the tool to work. The use of a user name is optional.

Result

The change in the logging level persists while the PolicyCenter application server is running only.

See also

- “system_tools command” on page 429

Set the log level using web services

Procedure

1. Ensure that the PolicyCenter application server is running.
2. Call the following method on the `SystemToolsAPI` web service:

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

Result

The change in the logging level persists while the PolicyCenter application server is running only.

See also

- *Integration Guide*

Set the log level from PolicyCenter

It is possible to set the log level temporarily from the PolicyCenter **Set Log Level** screen.

Procedure

1. Log into Guidewire PolicyCenter using an administrative account.
2. Navigate to the Server Tools **Set Log Level** screen.
3. Select a logging level from the drop-down list.
4. Enter the new logging level for that category.

Result

The change in the logging level persists only while the PolicyCenter server is running.

See also

- “The Set Log Level screen” on page 361

Configuring archive logging operations

About this task

Your PolicyCenter installation must support archiving in order for you to enable archive logging. To enable archiving logging in Guidewire PolicyCenter, do the following:

- Make the archiving-related logging code in file `log4j2.xml` active by removing the comments surrounding the example loggers.
- Configure the example archiving loggers as needed.

See also

- “Print details of the archive graph” on page 43
- “Generate a separate archive log” on page 43
- “Batch processes that generate archive data” on page 44
- *Configuration Guide*

Print details of the archive graph

About this task

As the PolicyCenter server starts, the server builds the archive domain graph. To print out the details of how PolicyCenter determines the graph, perform the following steps.

Procedure

1. Log into PolicyCenter Studio.
2. Navigate to the following location and open file `log4j2.xml` for editing.

   configuration→config→logging
3. Add the following code to the file.

   ```
   <Logger name="Server.Archiving.Graph" level="debug">
     <AppenderRef ref="Console"/>
     <AppenderRef ref="DailyLogFile"/>
   </Logger>
   ```

   Add either or both of the `<AppenderRef>` elements depending on where you want to print the information.
4. Rebuild and redeploy the PolicyCenter application.

Result

After restarting the application server, you see details in the console log of the archive domain graph.

See also

- *System Administration Guide*

Generate a separate archive log

About this task

Guidewire recommends as a general practice that you create a separate log file for archive information. To do so, perform the following steps.

Procedure

1. Log into PolicyCenter Studio.
2. Navigate to the following location and open file `log4j2.xml` for editing.

   configuration→config→logging
3. Add an archive `<RollingFile>` element similar to the following example to the file.

   ```
   <RollingFile name="ArchiveLog" fileName="${guidewire.logDirectory}/archiveLog.log"
         filePattern="${guidewire.logDirectory}/archiveLog.log%d{.yyyy-MM-dd}">
     <PatternLayout pattern="${file.defaultPattern}" charset="UTF-8"/>
     <TimeBasedTriggeringPolicy/>
   </RollingFile>
   ```
4. Add a `<Logger>` archive element that corresponds to the `<RollingFile>` archive element that you added in the previous step, for example:

   ```
   <Logger name="Server.Archiving" level="info">
     <AppenderRef ref="ArchiveLog"/>
   </Logger>
   ```
5. Rebuild and redeploy Guidewire PolicyCenter.

Batch processes that generate archive data

In Guidewire PolicyCenter, the following batch processes generate archive-related data:

- Archive Policy Term batch processing
- Retrieve Policy Terms batch processing

chapter 3

PolicyCenter business event logging

A *business event* is an event of special interest in the PolicyCenter application, such as a user logging into Guidewire PolicyCenter. PolicyCenter associates various types of information with each defined business event. For example, as a user logs into PolicyCenter, PolicyCenter captures the following information about this event:

- User name
- Generated session ID
- CSRF token

After you capture information about these events, it is possible to output the information in multiple ways. For example, you can use the captured information to do the following.

- Write the special event information to a log file and mark the event information so as to make it easy to find in the log and extract for additional use.
- Use the special event information extracted from the log to generate charts and dashboards in third-party tools such as SumoLogic.

Guidewire intentional logging

In Guidewire InsuranceSuite applications, *intentional logging* is the act of capturing information about specific business events or application metrics and outputting that information to the PolicyCenter application logs. A *marker* is a named, hierarchical, object that Guidewire uses to provide additional information in the PolicyCenter application logs. The use of a marker makes it relatively easy to find, parse, and extract the log record associated with a special event.

For example, the following sample log message contains the special `<WebLogin>` marker.

```
INFO User <WebLogin> User Login…
```

In this example, `User` is the name of a logging category. `<WebLogin>` is an event marker that signifies a user has logged into PolicyCenter.

Formatting business event information

Most business events have some associated data. For example, logging into Guidewire PolicyCenter generates the following types of data:

- User name
- Generated jsession-id
- CRSF token

To be useful to a log monitoring tool, the log needs to format business event data so that it is easily extractable. Depending on the monitoring service involved, the information needs to be formatted either as a key-value map or as a formatted string.

The following code is an example of a formatted string.

```
{user="su",  userId="3",  from="0:0:0:0:0:0:0:1",  elapsedTimeMs=23,
      external=true,  error="java.lang.RuntimeException: behavior code is Error"}
```

Notice that the code formats string values inside double quotes but does not use quotes with numbers and Boolean values. The code formats the exception as "*exception_class*: *exception_message*".

Performance

Guidewire intends for intentional logging to be enabled always. The use of intentional logging does not affect the performance of Guidewire InsuranceSuite applications.

Logging categories and logging markers

There are specific differences between PolicyCenter logging categories and SLF4J logging markers:

- Guidewire uses logging categories for component-based logging.
- Guidewire uses logging markers for event-based logging.

Component-based logging

Guidewire uses logging categories to manage and format multiple log messages produced by a particular software component. For example, this could be all log messages associated with rule execution.

Guidewire defines the various types of logging categories in public classes `PCLoggerCategory`. The class defines the logging categories as constants that you can access as `PCLoggerCategory.CONSTANT`. For example, you can access logging information associated with forms as `PCLoggerCategory.FORMS`.

For more information on logging categories, see “Logging in Guidewire PolicyCenter” on page 25.

Business event logging

Guidewire associates a single specific marker type with a specific business event. For example, Guidewire associates marker type `WebLogin` with a user logging into PolicyCenter.

Guidewire defines the SLF4J marker types in the following:

- Read-only class `PLLoggingMarker`
- Public class `PCLoggingMarker`

Each class defines the marker types as constants that you can access as `PLLoggingMarker.CONSTANT` or `PCLoggingMarker.CONSTANT`. For example, you can access a user log out event as `PLLoggingMarker.WEB_LOGOUT`.

For more information on logging markers, see “PolicyCenter business event logging” on page 45.

Configuring intentional logging

By default, Guidewire enables intentional logging in the base configuration of Guidewire PolicyCenter and intends for intentional logging to be active always.

There are two types of configuration possibilities for intentional logging in PolicyCenter:

- Enable or disable intentional logging globally for the entire PolicyCenter application cluster
- Control the individual application elements that PolicyCenter logs during intentional logging

Guidewire caches the intentional logging configuration on each cluster node. If you make a configuration change to intentional logging, PolicyCenter propagates the change to each node after some short period of time after the change. Guidewire expects that changes to intentional logging configuration are very infrequent. However, if you do make a change, PolicyCenter writes the change to the application logs.

ILElement elements

Guidewire abstracts the individual application elements that it logs in intentional logging as `ILElement` elements. Thus, an `ILElement` element is the atomic portion of a logging process that is subject to Guidewire intentional logging. An example of such an element in Guidewire PolicyCenter is the `QUOTE_SYNC` profile tag, which represents one step in the quoting process.

Global settings for intentional logging

Use `IntentionalLoggingEnabled`, a static Boolean property on `ILElement`, to enable or disable intentional logging globally. The following code illustrates how to set this value to either `true` or `false`.

```
ILElement.IntentionalLoggingEnabled = true
ILElement.IntentionalLoggingEnabled = false
```

By default, Guidewire enables intentional logging in the base configuration of Guidewire PolicyCenter and intends for intentional logging to be active always.

Individual settings for intentional logging elements

Use the Boolean `Enabled` property on specific `ILElement` elements to enable or disable individual elements in intentional logging. The following code illustrates this concept. The code enables intentional logging for `QUOTE_SYNC` element (`ILElement`) by setting the `ENABLED` property to `true`.

```
(PCProfilerTag.QUOTE_SYNC as ILElement).Enabled  = true
```

Interface ILElement

Use interface `ILELement` (in package `gw.api.intentionallogging`) to create logging markers programatically. Use the `ILELement` interface to create `ILElement` objects in static classes for use in logging code. See “Public methods on ILElement” on page 47 for information on the interface public methods.

The following code illustrates this concept.

```
class CustomILElements {
  ...
  public static final ILElement CUSTOM_ILELEMENT = ILElement.of(ILElementIdentifier.of(TC_MANUAL, "customName"),
        MarkerFactory.getMarker("userMarker"))
  )
  ...
}
```

Note: By default, any `ILElement` object that you create in code is active by default, as is intentional logging in general.

Then, to be useful, you must use the custom `ILElement` in Gosu code somewhere.

```
...
if(CustomElements.CUSTOM_ILELEMENT.Enabled) {
  logger.info(CustomILElement.CUSTOM_ILELEMENT.Marker, "some_meaningful_text")
}
...
```

You also need to configure an `<appender>` element in file `log4j2.xml` if you want to print the marker to a specific application log file.

Public methods on ILElement

Interface `ILElement` contains the following public methods.

`getIdentifier()`
: Method returns an `ILElementIdentifier` object for this `ILElement` object.

`getMarker()`
: Method returns the SLF4J marker associated with this `ILElement` object.

`setEnabled(enabled)`
: Method enables intentional logging for this particular `ILElement` object.

`isEnabled()`
: Method returns `true` or false depending on whether this particular `ILElement` object supports intentional logging.

`setIntentionalLoggingEnabled(enabled)`
: Method sets intentional logging globally for all `ILElement` objects.

`isIntentionalLoggingEnabled()`
: Method returns `true` or false depending on whether intentional logging is enabled globally.

`of(identifier, marker)`
: Method returns an `ILElement` object of the specified type, which depends on the given *`identifier`* (`ILElementIdentifier`) and *`marker`* (SL4J marker).

Class PLLoggingMarker

Internal Guidewire class `com.guidewire.pl.logging.PLLoggingMarker` defines the marker types that are valid in the base configuration. This class defines each marker type as a constant. Use the following syntax to reference each marker type:

```
PLLoggerMarker.MARKER_TYPE
```

For example, to reference the user logon marker type, use the following code:

```
PLLoggerMarker.WEB_LOGON
```

The following table lists the marker constants defined in class `PLLoggingMarker` that are publicly available for use.

Event type	Marker text
Web event	• `WEB_LOGON` • `WEB_LOGOUT` • `WEB_REQUEST`
Batch process event	• `BATCH_PROCESS_STARTED` • `BATCH_PROCESS_COMPLETED` • `WORK_QUEUE_WORK_ITEM_PROCESSED`
Web service event	• `WEB_SERVICE_REQUEST` • `WEB_SERVICE_OPERATION`
Message event	• `MESSAGE_BEFORE_SEND` • `MESSAGE_SEND` • `MESSAGE_AFTER_SEND` • `MESSAGE_ACK_RECEIVED`
Configuration service event	• `CONFIG_SERVICE_STATE`
REST API event	• `REST_REQUEST`

The marker types that Guidewire defines in class `PLLoggingMarker` are of type `Marker` from the SLF4J library. For more information, refer to the following SLF4J documentation:

```
http://www.slf4j.org/apidocs/org/slf4j/Marker.html
```

Public methods

Class `PLLoggingMarker` contains a single public method:

```
marker(String)
```

Use this method to create a custom name for a logging marker, for example:

```
PLLoggingMarker.marker("custom_text")
```

Class PCLoggingMarker

In addition to the base `PLLoggingMarker` class, Guidewire provides a `gw.api.system.PCLoggingMarker` class that defines a number of PolicyCenter-specific marker types. This class defines each marker type as a constant. Use the following syntax to reference each marker type:

```
PCLoggerMarker.MARKER_TYPE
```

For example, to reference a marker for the validation step in the quote process, use the following code:

```
PCLoggerMarker.QUOTE_VALIDATE
```

The following table lists the marker constants defined in class `PCLoggingMarker` that are publicly available for use.

Event type	Marker text
Quote processes	• QUOTE_SYNC • QUOTE_VALIDATE • QUOTE_MERGE • QUOTE_SEBMENTION • QUOTE_CHECK_UW_ISSUE • QUOTE_PREPARE_ACCOUNT_SYNCABLES • QUOTE_INFER_FORMS • QUOTE_DENORM_FINANCIALS • QUOTE_FINISH • QUOTE_HANDLE_RESPONSE • QUOTE_HANDLE_REINSURANCE • QUOTE_SETPAYMENTINFO

The marker types that Guidewire defines in class `PCLoggingMarker` are of type `Marker` from the SLF4J library. For more information, refer to the following SLF4J documentation:

```
http://www.slf4j.org/apidocs/org/slf4j/Marker.html
```

Public methods

Class `PCLoggingMarker` does not contain public methods.

Class LogMessageParams

PolicyCenter uses `LogMessageParams` utility objects to create and format key-value pairs associated with a business event. PolicyCenter uses the data object to format key-value pairs for better human readability and machine parsing.

The following log record is an example of a formatted marker string, enclosed by curly braces {…}.

```
<BatchProcessCompleted> Batch process completed {type="Workflow", processHistoryId=103,
      elapsedTimeMs=13, error=null}
```

Notice the following:

- String values have surrounding double quotes, `type="Workflow"`, for example.
- Boolean, number, and `null` values have no special formatting; `processHistoryId=103` or `error=null`, for example.
- Exceptions, if present, for example:

```
"Exception class: exception message"; error="java.lang.RuntimeException: Behavior code
is Error"
```

Guidewire uses these formatting conventions as it is easy to parse this information programmatically. Third-party tools, such as SumoLogic, can easily parse a log record that contains this type of formatted information.

Refer to the following SumoLogic web site for more information about SumoLogic:

`https://www.sumologic.com/`

Public methods on LogMessageParams

Class `LogMessageParams` contains the following public methods.

`create()`

This method returns a new `LogMessageParams` data utility object, for example:

`UserLogOn = LogMessageParams.create()`

`put(String, Object)`

This method adds the specified key-value pair to a `LogMessageParams` data utility object, for example:

```
UserLogOn.put("user", username)
         .put("userId", user.getID())
         .put("session", session.getId()))
```

`put(LoggingKeyValue)`

This method adds the key-value pair specified by a `LoggingKeyValue` object to a `LogMessageParams` data utility object. This action is similar to the action of the previous `put` method.

`toString()`

This method returns the formatted message to insert into the log record. For example, given the `UserLogOn` object defined previously, the following code…

`UserLogOn.toString()`

... returns a message string that uses the following format…

`{user="su", userId="3", session="qq0kis1ihdr11cq8r2ait576w"}`

Guidewire Profiler

In the base configuration, Guidewire integrates intentional logging with Guidewire Profiler. As a consequence, it is is possible to use class `LoggableProfilerTag` to create a profiler tag and link it to a defined logging marker. PolicyCenter then logs any code that contains one of these profiler tags using the Profiler `push` and `pop` methods.

This logging mechanism:

- Does not depend on whether Guidewire Profiler is enabled or disabled.
- Does not depend on any Profiler frame stacks.
- Does not require any change to profiling statements.

The relationship between profiler tag and PolicyCenter logging marker is not one-to-one relationship. It is possible to associate a single logging marker with multiple profiler tags.

Class LoggableProfilerTag

Use class `LoggableProfilerTag` to construct a Guidewire Profiler tag and associate the tag with a defined SLF4J marker. The logging mechanism then automatically logs the intentional logging marker as it encounter code that contains the profiler tag.

The following code illustrates these concepts.

```
uses gw.api.intentionallogging.ILElement
uses gw.api.profiler.LoggableProfilerTag
uses org.slf4j.MarkerFactory

...
var tag = new LoggableProfilerTag("UxTag", MarkerFactory.getMarker("UxMarker"))
...
```

Notice the following:

- The code uses class `LoggableProfillerTag` to create the profiler tag.
- The code associate the new profiler tag with a specific logging marker.

You also need to configure an `<appender>` element in file `log4j2.xml` if you want to print the marker to a specific application log file.

Traceability IDs

A *Traceability ID* is a `String` UUID (Universally Unique Identifier) that Guidewire uses to track a single user transaction or operation across multiple logging statements. The logging statements can span calls across multiple applications in inter-application integration. PolicyCenter stores the traceability ID string in a MDC (Mapped Diagnostic Context) object that the Log4j logging system maintains. The MDC object is essentially a key-value map that PolicyCenter stores in a `ThreadLocal` variable.

More information

Refer to the following documentation for more information on MDC objects and their usage:

`https://www.slf4j.org/api/org/slf4j/MDC.html`

`https://logback.qos.ch/manual/mdc.html`

The TraceabilityIDPlugin plugin

In the base configuration, the default implementation class for the `TraceabilityIDPlugin` plugin is Gosu class `DefaultTraceabilityIDPlugin`. This class provides three overloaded versions of the `withTraceabilityID` method. All of these methods provide a way to configure how to propagate traceability IDs at creation points indicated by the `creationPoint` iterator. See the JavaDoc for the `TraceabilityIDPlugin` interface for more information on these methods.

Method	Returns
`withTraceability(creationPoint, maybeCurrentTID)`	`TraceabilityID`
`withTraceability(creationPoint, maybeCurrentTID, message)`	`TraceabilityID`
`withTraceability(creationPoint, maybeCurrentTID, pluginName)`	`TraceabilityID`

The default length of a traceability ID produced by the base configuration plugin implementation is 36 characters. The maximum allowable length for a traceability ID is 64 characters. Any attempt to alter the plugin implementation to create an ID longer than 64 characters generates an error message in the application log.

Class TraceabilityUtils

Public class `TraceabilityUtils` contains useful utility methods for working with traceability IDs. The following table lists a few of these methods.

Method	Returns
`setTraceabilityID(traceabilityID)`	-
`getTraceabilityID()`	`String`
`removeTraceabilityID()`	-

The `setTraceabilityID`, `getTraceabilityID`, and the `removeTraceability` methods behave as one would expect, meaning:

- Method `setTraceabilityID` sets the traceability ID to the specified value.
- Method `getTraceabilityID` retrieves the value of the current traceability ID.
- Method `removeTraceability` deletes the value of the current traceability ID.

For example, to set the traceability ID for a specific transaction, you can explicitly create a new traceability ID using the following syntax:

```
    TraceabilityUtils.setTraceabilityID("tr-1-1111")
```

Configuring the traceability ID in log4j2.xml

Logging configuration file `log4j2.xml` contains several important elements that you need to configure to work with traceability IDs and markers.

The <Property> element

In the base configuration, Guidewire adds the following code to the `<Property name="file.defaultPattern">` element for both traceability ID and marker string formatting. This property defines the default formatting to use in building the names of the log file that PolicyCenter generates on a daily basis.

```
<Property name="file.defaultPattern">
      ${gw:serverId}
      %-8.24X{userID}
      %36.36X{traceabilityID}
      %d{yyyy-MM-dd HH:mm:ss,SSS}
      %p
      %notEmpty{&lt; %marker &gt;}
      %m
      %n
</Property>
```

The <Console> element

In the base configuration, Guidewire adds the following code to the `<Console>` element for both traceability ID and marker string formatting. The `<Console>` element defines how PolicyCenter sends logging messages to the application console.

```
<Console name= "Console" follow= "true">
  <PatternLayout pattern="${gw:serverId}
        %-8.24X{userID}
        %36.36X{traceabilityID}
        %d{yyyy-MM-dd HH:mm:ss,SSS}
        %p
        %notEmpty{&lt; %marker &gt;}
        %m
        %n"
        charset="UTF-8"/>
</Console>
```

Working with web event markers

This topic provides the examples of how to work with the following web event markers:

- “Web login” on page 52
- “Web logout” on page 53
- “Web request” on page 53

Web login

Example

The following code is an example of how to construct an entry for the application log that lists information associated with a user logging into Guidewire PolicyCenter.

```
PLLoggerCategory.USER.info(PLLoggingMarker.WEB_LOGIN, "User Login {}", LogMessageParams.create()
      .put("user", username)
      .put("userId", user.getID())
      .put("csrfToken", session.getAttribute(CSRFTokenUtils.CSRF_TOKEN_SESSION_ATTRIBUTE))
      .put("from", request.getRemoteAddr())
      .put("session", session.getId()));
```

In this code, notice the following:

- The use of the `PLLoggingMarker.WEB_LOGIN` marker to set the specific business event of interest.
- The use of the `LogMessageParams.create` method to create a new data utility object to store the key-value pairs.
- The uses of the `LogMessageParams.put` method to store multiple key-value pairs on the data utility object.

This code creates a message in the application log similar to the following:

```
User <WebLogin> User Login {user="su", userId="3",
      csrfToken="d41f5a4b2ce52c1594f34ebe457ec153545c8039",
      from="0:0:0:0:0:0:0:1", session="qq0kis1ihdr11cq8r2ait576w"}
```

Notice that the message text (`User Login`) duplicates the meaning of the logging marker (`<WebLogin>`) uses in the example.

Web logout

Example

The following code is an example of how to construct an entry for the application log that lists information associated with a user logging out of Guidewire PolicyCenter.

```
PLLoggerCategory.USER.info(PLLoggingMarker.WEB_LOGOUT, "User logout {}", LogMessageParams.create()
      .put("user", user.getCredential().getUserName())
      .put("userId", user.getID()));
```

Notice the use of the `PLLoggingMarker.WEB_LOGOUT` marker and the `create` and `put` methods in a similar fashion to the previous example.

This code creates a message in the application log similar to the following:

```
INFO User <WebLogout> User logout {user="su", userId="3"}
```

Web request

Example

The following code is an example of how to construct an entry for the application log that lists information associated with user activity on a Guidewire PolicyCenter application screen.

```
eventSource = ...

LocationStack frame = PLDependencies.getNavigator().getFrame(FrameId.MAIN)

LOG_CATEGORY.info(PLLoggingMarker.WEB_REQUEST, "runLifecycleSteps {}", LogMessageParams.create()
      .put(ElapsedTime.of(System.nanoTime() - _startTimeInNanos, TimeUnit.NANOSECONDS))
      .put("location", Joiner.on('/').join(frame.iterator()))
      .put("source", eventSource.toString()))
```

Notice the use of the `PLLoggingMarker.WEB_REQUEST` marker and the `create` and `put` methods in a similar fashion to the previous examples. However, in this example, also notice the use of the `PLDependencies.getNavigator().getFrame(FrameID.MAIN)` method to extract the name of the PCF page on which the web request was made.

This code creates a message in the application log similar to the following:

```
INFO <WebRequest> runLifecycleSteps {elapsedTimeMs=222,
      location="ServerTools/InfoPages/ArchiveInfo",
      source="ServerTools/MenuLinks/ServerTools_InfoPages/(LocationGroupMenuItemWidget)
            /InfoPages_ArchiveInfo"}
```

Working with batch process markers

This topic provides the examples of how to work with the following batch process markers:

- "Batch process start" on page 54
- "Batch process completion" on page 54
- "Work item processing" on page 55

Batch process start

Example

The following code is an example of how to construct an entry for the application log that tracks the start of each batch process that Guidewire PolicyCenter executes.

```
localProcess = ...

LOG.info(PLLoggingMarker.BATCH_PROCESS_STARTED, "Batch process started {}", LogMessageParams.create()
      .put("type", localProcess.getType().getCode())
      .put("processHistoryId", localProcess.getProcessHistoryId()))
```

In this code, notice the following:

- The use of the `PLLoggingMarker.BATCH_PROCESS_STARTED` marker to set the specific business event of interest.
- The use of the `LogMessageParams.create` method to create a new data utility object to store the key-value pairs.
- The uses of the `LogMessageParams.put` method to store multiple key-value pairs on the data utility object.

This code creates a message in the application log similar to the following:

```
INFO Server.BatchProcess <BatchProcessStarted> Batch process started {type="UserException",
      processHistoryId=1315}
```

Batch process completion

Example

The following code is an example of how to construct an entry for the application log that tracks the completion of each batch process that Guidewire PolicyCenter executes.

```
process = ...

LOG.info(PLLoggingMarker.BATCH_PROCESS_COMPLETED, "Batch process completed {}",
      LogMessageParams.create()
            .put("type", process.getType().getCode())
            .put("processHistoryId", process.getProcessHistoryId())
            .put(ElapsedTime.ms(process.elapsedTime(systemClock().systemTimeMillis())))
            .put("error", throwable))
```

Notice the use of the `PLLoggingMarker.WORK_QUEUE_WORK_ITEM_PROCESSED` marker and the `create` and `put` methods in a similar fashion to the previous example.

This code creates a message in the application log similar to the following:

```
INFO Server.BatchProcess <BatchProcessCompleted> Batch process completed
      {type="ProcessCompletionMonitor", processHistoryId=1314,
       elapsedTimeMs=19, error=null}
```

Work item processing

Example

The following code is an example of how to construct an entry for the application log that tracks the processing of a work item in Guidewire PolicyCenter..

```
_itemLog.info(PLLoggingMarker.WORK_QUEUE_WORK_ITEM_PROCESSED, "Work item processed {}",
      LogMessageParams.create()
            .put("type", getQueueType().getCode())
            .put(ElapsedTime.ms(workElapsed))
            .put("error", wiException)
            .put("failed", failed))
```

Notice the use of the `PLLoggingMarker.WEB_LOGOUT` marker and the `create` and `put` methods in a similar fashion to the previous example.

If the processing of the work item was successful, this code creates a message in the application log similar to the following.

```
INFO Workqueue.Item <WorkItemProcessed> Work item processed {type="UserException",
      elapsedTimeMs=3, error=null, failed=false}
```

It the processing of the work item was not successful, this code creates a message in the application log similar to the following if PolicyCenter.

```
INFO Workqueue.Item <WorkItemProcessed> Work item processed {type="TestWorkQueue",
      elapsedTimeMs=24, error="java.lang.RuntimeException: behavior code is Error",
      failed=true}
```

Working with web service request markers

This topic provides the examples of how to work with the following web request markers:

- “Web service request” on page 55

Web service request

Example

The following code is an example of how to construct an entry for the application log that lists information associated with a user making a request in a PolicyCenter screen.

```
getILogger().info(PLLoggingMarker.WEB_SERVICE_REQUEST, "WS request processed {}",
      LogMessageParams.create()
            .put("name", requestElement == null? null: requestElement.get$QName())
            .put("from", userAddr)
            .put("user", userName)
            .put(ElapsedTime.ms(timeMs))
            .put("error", error));
```

In this code, notice the following:

- The use of the `PLLoggingMarker.WEB_SERVICE_REQUEST` marker to set the specific business event of interest.
- The use of the `LogMessageParams.create` method to create a new data utility object to store the key-value pairs.
- The uses of the `LogMessageParams.put` method to store multiple key-value pairs on the data utility object.

This code creates a message in the application log similar to the following:

```
INFO XML.Runtime <WebServiceRequest> WS request processed {
      name="{http://guidewire.com/pl/ws/gw/wsi/pl/SystemToolsAPI}
      getClusterState", from="127.0.0.1", user=null, elapsedTimeMs=71, error=null}
```

part 2

Server administration

chapter 4

PolicyCenter server configuration

This topic discusses ways to set up your PolicyCenter server environment.

Important PolicyCenter server configuration files

In PolicyCenter Studio™, Guidewire places several important configuration files at the root of following **Project** directory:

configuration→config

If you do not see this directory path, expand the Studio **Project** folder. The following table describes these important files.

File	Description
`config.xml`	File `config.xml` contains global system parameters that you use to control the behavior of Guidewire PolicyCenter. These configuration parameters govern large-scale system options, such as authentication, server clustering, and the business calendar.
`database-config.xml`	File `database-config.xml` stores database connection information and Data Definition Language (DDL) options.

Internal configuration files

Directory **configuration→config** also contains internal file `config-service.yml`. This file is for Guidewire internal use only. Do not delete this file or modify it in any way.

See also

For more information on file `config.xml` and basic application configuration, see the following:

- "The Configuration screen" on page 362
- *Configuration Guide*

For more information on file `database-config.xml` and basic database configuration options, see the following:

- "Database configuration" on page 237
- "Database maintenance" on page 269
- *Installation Guide*

Understanding the PolicyCenter server environment

During startup, PolicyCenter calculates key environment and server properties. These property values describe the environment in which the server runs. After you set environment properties, you can access these properties through the following methods:

- PolicyCenter commands
- Gosu code
- Java code

It is possible to specify environment property values and server roles either through JVM (Java Virtual Machine) options or through the use of the `<registry>` element:

- You define environment and server properties in the `<registry>` element in file `config.xml`.
- You set environment and server properties through command prompt JVM options as you to start a PolicyCenter server.

IMPORTANT Guidewire recommends that you specify server IDs and server roles for individual application servers using JVM options as you start the application server rather than defining these values using the `<registry>` element in `config.xml`.

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 configuration file `config.xml` to specify and control one or multiple PolicyCenter server environments.

See also

- "Important PolicyCenter server configuration files" on page 59
- "Understanding the configuration <registry> element" on page 60
- "JVM options and server properties" on page 64
- "Configuration parameters by environment" on page 67
- *Installation Guide*

Understanding the configuration <registry> element

It is possible to set the PolicyCenter server environment using the `<registry>` element in `config.xml`. Through the `<registry>` element, you can define the following:

- The set of server roles that are valid for this cluster.
- The set of server roles that are valid for each individual server instance in the cluster.
- The environment or environments in which a specific server operates, a development environment and/or a production environment, for example.

Also through the `<registry>` element, you can redefine certain system properties that you specify at server startup.

IMPORTANT Guidewire recommends that you specify server IDs and server roles for individual application servers using JVM options as you start the application server rather than defining these values using the `<registry>` element in `config.xml`.

See also

- "The <registry> element" on page 60
- "Example syntax for the <server> element" on page 62
- "Defining a new server role" on page 64

The <registry> element

The `<registry>` element in file `config.xml` has the following syntax.

```
<registry roles="role1, role2, …" >
  <server env="environment1, environment2, …" roles="role1, role2, …" serverid="serverID" />
  <systemproperty name="name" value="value" default="default" />
</registry>
```

File `config.xml` contains exactly one required `<registry>` element. The `<registry>` element can contain zero to many `<server>` and `<systemproperty>` elements.
The attributes on the various elements have the following meanings.

Element	Attribute	Required	Description
`registry`	`roles`	Yes	List of valid cluster roles. See "Server roles" on page 169 for a description of the default server roles that Guidewire provides in the base configuration.

`server`	`env`
	`serverid`
	`roles`
		No	The `<server>` attribute entries are legacy and not required. These attributes have the following meanings:
			• `env` - The name of the environment in which the server operates. The default is `null`. The value of env is immutable while the server is running.
			• `serverid` - Unique server name of a server instance in the cluster. If you do not specify this value explicitly, PolicyCenter sets it to the `hostname` of the PolicyCenter server.
			• `roles` - Specifies role or roles to assign to the server specified by `serverid`.
			It is a better practice to set these server attributes using JVM options during server startup. For example, the following JVM option sets both the server ID and server roles:
			`-Dserverid="dev2#messaging,scheduler"`
			Use the # syntax to indicate the following text is a server role and not a server ID.

`systemproperty`	`name`	Yes	Specifies the name of the Java `-D` system property that you want to redefine. This value can be one of the following: • `env` • `serverid` If you run multiple server instances on the same host machine, define a `serverid` value for each server instance with a separate `systemproperty` entry.
	`value`	Yes	New name of the system property.
	`default`	Yes	Default property value to use if you do not specify a value at the command prompt.

The following code shows an example `<registry>` element.

```
<registry roles="batch, scheduler, workqueue, messaging, startable, ui, custom1">
  <systemproperty name="env" value="my.env" default="production"/>
  <systemproperty name="serverid" value="my.id" default="myDefault"/>
</registry>
```

Notice that:
- The set of valid server roles includes a custom role (`custom1`).
- There are two system property redefinitions, one for `env` and one for `serverid`.

Property serverid.noroles

In standard usage, the value of property `serverid` includes also the list of server roles defined for that server. If you want to use this property in code, without the list of server roles, use `serverid.noroles` instead. For example, instead of using `gw.pc.serverid`, use `gw.pc.serverid.noroles` to suppress the list of server roles associated with this server ID.

Environment and logging

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`*`-log4j2.xml` file. If this file does not exist or if `env` is `null`, then the logging configuration is taken from the default `log4j2.xml` file.

Creating special server roles

It is possible to define new server roles by adding the role to the list of roles specified by the `roles` attribute on the `<registry>` element. For an example of how to define and use a specialized server role, see "Work queues and server roles" on page 127.

Accessing system properties in code

Gosu class `gw.api.system.server.ServerUtil` contains methods for working with system properties associated with servers. See the *Integration Guide* for information on how to use this library.

See also

- "Server roles" on page 169

Example syntax for the <server> element

The following example illustrates how to use the `<server>` element of the `<registry>` element in `config.xml` to set various server properties.

```
<registry roles="...,...,...,...">
  <server env="..." serverid="..." roles="..." />
</registry>
```

Use the `roles` attribute to list the valid server roles for your installation. However, the existence of the `<server>` element is a legacy artifact. Guidewire discourages the use of the `<server>` element. Instead, use JVM options to set these values.

Using a JVM option to set server properties

Guidewire recommends that you set server system properties using a `-D` JVM option from a command prompt at server startup. For example, you can use JVM options to set the environment variable (`env`) and the server ID and server roles (`serverid`) at runtime using the syntax listed in the following table. The option syntax varies by server type. Notice the entries for `serverid`. The use of a starting # mark indicates that the following text is a server role and not a server ID.

Server type	Syntax	Example

QuickStart (Jetty)	`-Denv=...`
	`-Dserverid=...`
		`gwb runServer -Denv=test2`
		`gwb runServer -Dserverid=t2#messaging`

Tomcat	`-Dgw.pc.env=...`
	`-Dgw.pc.serverid=...`
		`-Dgw.pc.env=test2`
		`-Dgw.pc.serverid=t2#messaging`

If you are starting the Jetty development server in Guidewire Studio™ for PolicyCenter, use the syntax for Tomcat in the **Run - Server** configuration dialog, for example:

```
-Dgw.pc.serverid=testServer
```

For more information, see "Start the application server from Guidewire Studio for PolicyCenter" on page 82.

How PolicyCenter uses a -D JVM option

The values of `env` and `serverid` are immutable while the server is running.

PolicyCenter determines the value of a `-D` option in the following manner, using `-Dserverid` (on Jetty) as an example:

- If you specify a `-Dserverid=prodserver` JVM option at the command prompt at server startup, PolicyCenter sets the value of `serverid` for that server to `prodserver`.
- If you do not specify a `-Dserverid` JVM option at server start, PolicyCenter checks the server registry for a `serverid` value defined by a `server` entry. If found, PolicyCenter uses that value. In the example, the `serverid` value is `prodserver`.
- If you do not specify the JVM option, and no `serverid` value defined by a `server` entry exists, PolicyCenter sets `serverid` to the host name of the computer. Under some extreme security settings, this value is not available, in which case PolicyCenter sets the `serverid` to `localhost`.

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

Viewing server information in PolicyCenter

PolicyCenter shows the `serverid` and `role` values for each server in a PolicyCenter cluster on the Server Tools **Cluster Members and Components** screens. You must log in as an administrative user to access the Server Tools.

See also

- "Understanding the configuration <registry> element" on page 60
- "JVM options and server properties" on page 64
- "The Cluster Members screen" on page 391

Example syntax for the <systemproperty> element

The following example illustrates how to set the server environment using the `<systemproperty>` element of the `<registry>` element in `config.xml`.

```
<registry roles="…" >
  <systemproperty name="env" value="my.env" default="production"/>
</registry>
```

Notice that the defined `<systemproperty>` element:

- Redefines the name of the `env` property to be `my.env`. Thus, in places in which you formerly used `env`, you now use `my.env`. For example, `-Denv` becomes `-Dmy.env`.
- Sets a default value (`production`) for the system property.

To use this system property, you specify its value as a `-D` JVM option at server startup. The option syntax varies by server type.

QuickStart (Jetty)	`-Dmy.env=…`
Tomcat	`-Dgw.pc.my.env=…`

The value of `my.env` is immutable while the server is running.

PolicyCenter determines the value of the redefined `env` system property in the following manner, using `-Dmy.env` (on Jetty) as an example:

- If you specify the `-Dmy.env=test` JVM option at the command prompt at server startup, PolicyCenter sets the value of `env` to `test`.
- If you do not specify a `-Dmy.env` option at server start, PolicyCenter sets `env` to the value of `default` that you specified in the `systemproperty`, in this example, `production`.
- If you do not set the value of `my.env` either through a default `registry` property or with a JVM option, PolicyCenter sets the value of the `env` property to `null`.

PolicyCenter ignores any attempt to set the environment property through the command prompt if you have previously defined that property using a `<systemproperty>` element. For example, suppose that you set the following `<registry>` element in `config.xml`:

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

Then, at server startup, you specify a `-Denv="test"` JVM option. PolicyCenter ignores any `-Denv` option that you specify on the command prompt and sets the `env` value to `standalone`.

See also

- "Understanding the configuration <registry> element" on page 60
- "JVM options and server properties" on page 64

Defining a new server role

You define server roles using the `roles` attribute on the `<registry>` element in file `config.xml`. In the base configuration, Guidewire defines the following server roles:

```
<registry roles="batch, workqueue, scheduler, messaging, startable, ui" />
```

To add a specialized server role, say, one to use in managing activities, you need merely to add the new server role to the list of roles:

```
<registry roles="batch, workqueue, activity, scheduler, messaging, startable, ui" />
```

See also

- "Defining a new work queue role" on page 127

JVM options and server properties

It is possible to set certain PolicyCenter server properties by using `-D` JVM options on the PolicyCenter `gwb` build commands. Some of these JVM options work with all the `gwb` build commands. Other JVM options work only with the `gwb runServer` build command. In all cases, the exact manner in which you define and use the JVM option depends on the type of server involved.

Once set, a server property is immutable while the server is running.

See also

- "The <registry> element" on page 60
- "Example syntax for the <server> element" on page 62
- "Example syntax for the <systemproperty> element" on page 63

JVM options for gwb build commands

The following JVM options work with many of the PolicyCenter `gwb` build commands.

JVM option syntax	Description
`-Denv=`*aaaa*	Starts the PolicyCenter server using the specified environment variable (env).

JVM option syntax	Description
`-Dgw.passthrough.`*`systemProperty=dddd`*	Starts the PolicyCenter server using the provided system property value. The option parameters have the following meanings: • *`systemProperty`* – The name of a system property defined in the `<registry>` element in file `config.xml`. • *dddd* – The value of the system property.

JVM env string value always lower case

If you use a JVM option to set the server environment, PolicyCenter converts the provided string to lower case automatically. For example, the conversion to lower case occurs if you use the following JVM options:

- Jetty - `-Denv`
- Tomcat - `-Dgw.pc.env`

Thus, if you enter `-Dgw.pc.env=TEST`, PolicyCenter converts the string `TEST` to the string `test`. Be aware that the automatic conversion of the `env` string to lower case can cause a referenced `env` value to not match the defined environment name.

Reference table of core gwb commands

It is possible to use the `-Denv` and the `-Dgw.passthrough` JVM options with some, but not all, of the `gwb` build commands. The following table indicates whether the listed JVM command options work with the core `gwb` commands (tasks).

Core task	Can use JVM option	Cannot use JVM option
`clean`		●
`cleanIdea`		●
`codegen`		●
`compile`		●
`dropDb`	●	
`genDataDictionary`	●	
`idea`		●
`runServer`	●	
`stopServer`	●	
`studio`		●

JVM option examples

For example, to pass `-D`*`mySystemProperty=someValue`* to build command `dropDB`, use the following command option.

```
gwb dropDb -Dgw.passthrough.mySystemProperty=someValue
```

Then, to make the `dropDB` command specific to a test environment, use the following command (for a Jetty server).

```
gwb dropDb -Denv=test -Dgw.passthrough.mySystemProperty=someValue
```

The following build command illustrates the use of a pass through system property.

```
gwb runServer -Dgw.passthrough.javax.net.ssl.keyStore=/Java/jdk1.8.0_74/jre/lib/security/
cacerts
```

See also

- "Setting JVM options in PolicyCenter" on page 66

JVM options specific to the runServer build command

The following JVM options work with the PolicyCenter `gwb runServer` command only.

JVM option syntax	Description
`-Dgw.port=`*nnnn*	Starts the PolicyCenter server on the specified port (*nnnn*). The server URL reflects this port number, for example: `localhost:`*nnnn*`/pc/PolicyCenter.do`
`-Dgw.server.mode=`*xxxx*	Starts the PolicyCenter server in the specified server mode. Valid values are: • `dev` • `prod` The default is `dev`.
`-Dserverid=`*aaaa* `-Dserverid=`*aaaa#bbbb*	Sets the server ID, and possibly, one or more server roles for the PolicyCenter server: • Server ID – Without the hash mark (`#`), *aaaa* represents a server ID only. • Server role – With the hash mark, `#`*bbbb* assigns the *bbbb* server role to the server with *aaaa* server ID. Use a comma-separated list, with no spaces, to list multiple server roles.

The exact JVM syntax to use depends on the server type, for example:

- Quickstart (Jetty) – Use `-Dserverid=`*aaaa*
- Tomcat – Use `-Dgw.pc.serverid=`*aaaa*

See also

- "Example syntax for the <server> element" on page 62
- "Setting JVM options in PolicyCenter" on page 66

Setting JVM options in PolicyCenter

It is possible to set server system properties using the `-D` JVM option syntax. How you set a command option depends on the server type:

JBoss	Pass the options as arguments to the JBoss `run` script.
QuickStart (Jetty)	Set the options at server start using the following syntax: `gwb runServer -Denv=…` `gwb runServer -Dserverid=...`
Tomcat	Set the options using the `CATALINA_OPTS` environment variable. Use the following syntax: `-Dgw.pc.env=…` `-Dgw.pc.serverid=...`
WebLogic	Edit the `startManagedWebLogic` file.
WebSphere	Open the **Administrative Console** and add the option to **Generic JVM arguments**. If you have multiple servers, set the option for each server.

See also

- "JVM options for gwb build commands" on page 64
- "JVM options specific to the runServer build command" on page 66

Configuration parameters by environment

Typically, you need to support more than one server environment. Guidewire recommends that you maintain at least the following environments:

- Development
- Test
- Deployment (production)

So that you do not have to change configuration parameters each time you switch between environments, PolicyCenter provides the ability to set configuration parameters for a specific environment or for a specific server or server role.

Setting the env attribute on <param>

Environment-specific parameters can reference environment properties to indicate in which environment or environments they are valid. You specify the environment or environments for a configuration parameter in file `config.xml` by adding an *`env`* attribute to the parameter definition. If you want a parameter to apply for multiple values of the `env` attribute, you can add a comma-separated list of values for the attribute.

Use the following syntax to add an *`env`* attribute to a parameter definition:

```
<param name="..." value="..." env="..."/>
```

Setting the server attribute on <param>

Although it is possible for the `server` attribute to specify a server ID value, the use of a server ID is usually too specific to be meaningful in most cases. Thus, Guidewire recommends that you set the value of the `server` attribute to one or more server roles instead. As multiple servers can share the same server role, the `server` attribute can affect any server with that server role. You can only set a single value for the `server` attribute, however.

Use the following syntax (notice the beginning hash mark) to specify a server role rather than a server ID for the `server` attribute:

```
server="#server_role"
```

Thus, `#batch` specifies the `batch` server role.

Note:

Typically, the only real need to use a server ID for the `server` attribute is if there is a host that contains multiple NICs (Network Interface Cards). In that case, it is possible to specify a URL with an address that is specific to single NIC on the host machine.

Setting configuration parameters at runtime

To have PolicyCenter use a specific version of a parameter, specify the environment in a JVM option at server startup, for example:

```
gwb runServer -Denv=test
gwb runServer -Dserver=#batch
```

Interactions between env and server values

Suppose that you define three environment-specific parameters with at most either a `server` attribute value or an `env` attribute value but not both. Thus:

- The first version of the parameter contains a `server` attribute value only.
- The second version of the parameter contains an `env` attribute value only.
- The third version of the parameter contains neither a `server` attribute value nor an `env` attribute value.

PolicyCenter processes server startup options against the parameter versions in the following manner:

- If the runtime `env` value resolves to a value that the parameter `env` attribute expressly specifies, PolicyCenter uses that version of the configuration parameter.
- If the runtime `env` value does not resolve to the `env` value of a configuration parameter, PolicyCenter applies the first corresponding parameter setting in which the runtime `server` attribute resolves to a known value.
- If neither of the two attributes resolves to known values, PolicyCenter applies the parameter setting corresponding to the parameter with neither a `server` attribute value nor an `env` attribute value.

The following code provides multiple definitions for the same configuration parameter.

```
<param name="BusinessDayStart" value="7:00 AM" env="test, test2, …" />
<param name="BusinessDayStart" value="8:00 AM" server="#batch, ui" />
<param name="BusinessDayStart" value="9:00 AM"/>
```

Keep these parameter definitions in mind as you consider the following examples.

Example 1

Suppose that you start the server with the following command:

```
gwb runServer -Denv=test
```

In this case, PolicyCenter starts the application server and sets the value of configuration parameter `BusinessDayStart` to 7:00 a.m.

Example 2

Suppose that you start the application server and set both the environment and the server role to the following values:

```
gwb runServer -Denv=test -Dserver=#batch
```

In this case, PolicyCenter still sets `BusinessDayStart` to the 7:00 a.m. value as the environment value is the controlling factor.

Example 3

Suppose that you start the application server and set the `server` value only:

```
gwb runServer -Dserver="#batch"
```

In this case, PolicyCenter sets `BusinessDayStart` to the 8:00 a.m. value.

Example 4

Suppose that start the application server and set neither an `env` value or a server role (`server`) value. In this case, PolicyCenter sets `BusinessDayStart` to the 9:00 a.m. value.

Example 5

Suppose that you define a parameter with both `env` and `server` attribute values. In such a case, PolicyCenter applies the corresponding parameter setting if both the `env` and the `server` attributes resolve to known values. For example, specify the `BusinessDayStart` parameter as follows:

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

Consistent with this code, PolicyCenter sets `BusinessDayStart` to 9:00 a.m. if `env` resolves to `test` and `server` resolves to `ui`. Otherwise, PolicyCenter sets `BusinessDayStart` to 8:00 a.m.

> **Note:** You cannot use multiple *<param>* elements that share common values for both the `server` and `env` attributes. The value or values of at least one of the attributes must differ.

For a list of configuration parameters, including information about which parameters can be set by environment, see the *Configuration Guide*.

See also

- “Understanding the configuration <registry> element” on page 60
- “JVM options and server properties” on page 64

Default configuration values and environment properties

At startup, PolicyCenter requires many parameters. Consider this carefully if you specify parameters by environment. In some cases, you might want to specify a configuration 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 to a known value if the other two parameter lines do not resolve:

```
<param name="ClusteringEnabled" value="false" server="batch, ui, workqueue" />
<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.

See also

- “Understanding the configuration <registry> element” on page 60
- “Configuration parameters by environment” on page 67
- “JVM options and server properties” on page 64

chapter 5

External server configuration

Guidewire PolicyCenter provides the means to modify certain configuration files without the necessity to rebuild and redeploy a server WAR or EAR file. The process involves inserting placeholders into the selected configuration files. PolicyCenter then reads a substitution value for the placeholder from an external configuration file rather from the configuration file packaged with the WAR/EAR file.

This mechanism provides a way to modify a well-defined portion of the PolicyCenter application configuration from outside the deployment WAR/EAR file. For example, it is possible to insert a placeholder into a PolicyCenter configuration file and then replace that value externally to modify URLs that access integration points.

About external server configuration

External server configuration contains the following core elements:

- Placeholders in configuration files.
- A plugin that determines the substitution value for placeholder replacement using, in part, a key embedded in the placeholder.
- A Substitutor process that manages the steps in the substitution process.

Configuration placeholders

It is possible to insert placeholders into configuration files that support external configuration. PolicyCenter reads the substitution values from a source external to the application WAR or EAR file and replaces the entire placeholder in the configuration file with the new value.

External configuration provider plugin

Plugin `ExternalConfigurationProviderPlugin` determines the substitution value of the configuration placeholder. Given information about the placeholder, the plugin returns the substitution value for the placeholder. In the base configuration, PolicyCenter provides a default implementation class for this plugin. However, it is also possible to create your own custom version of this plugin.

Substitutor process

A PolicyCenter internal Substitutor process manages the actual substitution process for external server configuration. The Substitutor process performs the following tasks:

- It parses the configuration files that support externalized configuration and extracts the embedded placeholders from those files.
- It extracts a key from each placeholder that it uses to identify each placeholder.
- It queries the external configuration provider plugin for the substitution value for each placeholder.
- It replaces the placeholder in the configuration file (in its entirety) with the value returned by the plugin.

It is not possible to create your own custom Substitutor process.

Placeholders

Guidewire uses the following syntax for the placeholder in a configuration file:

```
${key:defaultValue}
```

In this string:

- *`key`* is an identifier that the Substitutor uses, in part, to query the `ExternalConfigurationProviderPlugin` plugin for the substitution value that replaces the placeholder.
- *`defaultValue`* is an optional default value to use for substitution if the query of the `ExternalConfigurationProviderPlugin` plugin does not return a value.

Separate the key and its default value with a colon. The initial curly brace `{` must follow the initial `$` character directly, meaning that there cannot be a space between the `$` and the `{` brace.

As PolicyCenter encounters a placeholder in a configuration file:

- If the `lookupValue` method returns a value, PolicyCenter replaces the entire placeholder string with the returned value.
- If the `lookupValue` method does not return a value PolicyCenter uses the provided default value in the placeholder, if the default value exists.

Placeholder syntax

A placeholder must contain the following types only:

- Letters
- Digits
- Dots
- Underscores

As you construct a key, observe the following rules:

- A key cannot contain spaces.
- A key cannot start with a digit.
- A key cannot be an empty string (`${ }`).
- A key cannot contain special characters, such as $.

Also keep in mind:

- A default value is optional.
- A placeholder in an XML configuration file cannot span multiple nodes.
- A placeholder in a Java property file cannot span multiple properties.

Placeholder examples

The following are all examples of valid placeholders.

`${key}`

The placeholder contains a key value only.

`${key:}`

The placeholder contains an empty default value.

`${`*`key`*`:`*`defaultValue`*`}`

The placeholder contains both a key value and a default value.

`${`*`key`*`:`*`onlyFirstColon:counts:all:further:colons:are:part:of:default:value`*`}`

The placeholder contains a default value with multiple colons. The first colon indicates the beginning of the default string. Subsequent colons are part of the default value. For example, this can be a database URL in file `database-config.xml`.

`${`*`keyW1thAD1g1t`*`}`

The placeholder contains a key that contains a digit.

Files that support configuration substitution

The following PolicyCenter configuration file groups all support configuration substitution.

Configuration file group	namespace
`config.xml`	`config` (except for the registry node in this file)
`database-config.xml`	`database`
`inbound-integration-config.xml`	`inbound-integration`
`messaging-config.xml`	`messaging`
`scheduler-config.xml`	`scheduler`
`suite-config.xml`	`suite`
`*.gwp` (Plugin registry definition files)	`plugin`

Namespace

Each group of files that Guidewire supports for external configuration has a unique namespace. This namespace is a hard-coded string that uniquely identifies a given file group. A file group can consist of a single member, `config.xml`, for example. Or, a file group can contain multiple members, the set of all `*.gwp` files, for example. The Substitutor process infers the file group namespace from the file that it is parsing.

The Substitutor uses the namespace (along with the placeholder key) as a composite argument as it queries the `ExternalConfigurationProviderPlugin` plugin for the substitution value for the placeholder. Because each key is unique, it is possible to reuse a key in different file groups. For example, one can use the same URL key in file `database-config.xml` and in file `myPlugin.gwp` as the namespace for each is unique.

The external substitution file

In addition to adding placeholders to configuration files, you must create and populate the external file that holds the values for the placeholders. The external substitution file holds only single line entries of name/value pairs.

Populating the substitution file

Each group of configuration files that support substitution has a unique namespace value. You must incorporate the file group namespace into each name/value entry in the external substitution file. Use the following syntax for each entry in this file.

`namespace.key=substitutionValue`

You can give the external substitution file any name that you want.

Example

For example, file `database-config.xml` contains a `<dbcp-connection-pool>` element with a `jdbc-url` attribute. Suppose that you want to provide a value at server start for this URL. Creating a placeholder for a configuration file is a multi-step process:

1. Replace the `jdbc-url="..."` attribute string in file `database-config.xml` with `${jdbc_url}`. Thus, the `jdbc-url` entry in `database-config.xml` looks similar to the following example:

   ```
   <dbcp-connection-pool jdbc-url=${jdbc_url} />
   ```

2. Make an entry for the `jdbc-url` placeholder in the external substitution file that looks similar to the following entry:

   ```
   database.jdbc_url="someURL"
   ```

After the Substitutor process makes the substitution for the placeholder, the `<dbcp-connection-pool>` element looks similar to the following example:

```
<dbcp-connection-pool jdbc-url="someURL" />
```

Identifying the substitution file

Set the path to substitution file using the following system property command as you start the server:

```
gw.config.external.property.file=filename
```

However, to use this system property with the `gwb runServer` command, you must use the following construction:

```
gwb runServer -Dgw.passthrough.key=value
```

The following example illustrates this concept.

```
gwb runServer -Dgw.passthrough.gw.config.external.property.file="C:\Guidewire
\10.0\cc10_1853\modules\configuration\etc\ext-properties.properties"
```

Additional configuration file replacements

It is also possible to substitute values, in whole or in part, in the following configuration files.

Type	File	Replace what?	More information
Logging property file	`log4j2.xml`	Entire file	"Logging property files" on page 74
Free-text search engine configuration files	`solrserver-config.xml`	Placeholder	"Free-text search engine configuration files" on page 76
Web service collection files	`*.wsc`	Override URL	"Web service collection files" on page 75

Logging property files

Guidewire supports the substitution of the entire logging configuration file `log4j2.xml` by using a system property to specify the full path to an alternate `log4j2.xml` file. To make this change, add the following system property to the server start command:

```
gw.config.external.logging.file
```

Example of logging property file replacement

The following example replaces the `log4j2.xml` file packaged with the installation WAR/EAR file with a logging file that exists on a local machine:

```
gwb startServer -Dgw.config.external.logging.file="C:\tmp\loggingfile\MyLoggingFile.xml"
```

Web service collection files

Guidewire supports the use of `<override-url>` nodes in web service collection files (`.wsc` files) as a way to override the defined WSDL URL. To work with overridden URLs, use the following items:

- Public interface `IWsiWebserviceConfigurationProvider`
- Public API `gw.external.configuration.SubstitutionProperties`
- Property `ServerOverrideUrl` on the `WsdlConfig` object.

It is a common practice to set or override specific values of the `WsdlConfig` object, such as a sensitive user name and password. Property `ServerOverrideUrl` is simply another field on the `WsdlConfig` object.

For each web service, it is possible to create a custom implementation of interface `IWsiWebserviceConfigurationProvider` that binds to a WSC file through the `<configuration-provider>` node. Thus, in your implementation of `IWsiWebserviceConfigurationProvider`, it is possible to use API method `SubstitutionProperties.lookupValue` to retrieve the override URL then set the `WsdlConfig.ServerOverrideUrl` property to this new value.

Example file testClient.wsc

Suppose that you want the ability to provide a WSDL URL for use in a production server environment and a separate WSDL URL for use in all other server environments. Accordingly, you set up file `testClient.wsc` in a similar fashion to the following code:

```
<webservice-collection ... >
  ...
  <settings>
    <configuration-provider>
      <type-name>gw.util.TestConfigurationProvider</type-name>
    </configuration-provider>

    <override-url>
      <env>prod</env>
      <url>http://someUrl</url>
    </override-url>

    <override-url>
      <url>url_to_be_rewritten</url>
    </override-url>

  </settings>
   ...
</webservice-collection>
```

Notice the following about this code:

- The `<configuration-provider>` node binds class `TestConfigurationProvider` to this WSC file. Class `TestConfigurationProvider` manages the URL substitution process.
- The code provides for two override URLs. One URL is active in a production (`prod`) environment. The other URL is active in all other environments.

Example file TestConfigurationProvider

The following code shows an implementation of class `gw.util.TestConfigurationProvider`. Remember that `testClient.wsc` binds this class to the WSC file using the `<configuration-provider>` node.

```
package gw.util

uses gw.external.configuration.SubstitutionProperties
uses gw.xml.ws.IWsiWebserviceConfigurationProvider
uses gw.xml.ws.WsdlConfig
uses javax.xml.namespace.QName

class TestConfigurationProvider implements IWsiWebserviceConfigurationProvider {
  override function configure(serviceName: QName, portName: QName, config: WsdlConfig) {
    config.Guidewire.Authentication.Username = "..."
    config.Guidewire.Authentication.Password = "..."
```

```
      //Access the plugin through the public API
      var newValue = new SubstitutionProperties().lookupValue("ServiceName", "urlForTestWsc")

      config.ServerOverrideUrl = newValue
    }
  }
```

Notice the following in this code:

- The `ConfigurationProvider` class implements the `IWsiWebserviceConfigurationProvider` interface.
- The class uses the `lookupValue` method on the `SubstitutionProperties` API to retrieve the URL override value.
- The `lookupValue` method parameter *`ServiceName`* is a placeholder for an actual web service name.
- The `ServerOverrideUrl` statement writes the new URL to the non-production `<overrideUrl>` node in file `testClient.wsc`.

Free-text search engine configuration files

The free-text search engine `batchload.bat` (`batchload.sh`) script launches a Java program that uses file `solr-config.xml` to identify where and how the free-text search engine is running. It is possible to use configuration placeholders in file `solr-config.xml` in a similar manner to other Guidewire configuration files. This means that you can insert configuration placeholders into file `solr-config.xml` and use a separate configuration file to replace the original placeholder. The substitution file contains only single-line entries of the following type:

```
solr.key=placeholderValue
```

Set the path to the substitution file using the `gw.config.external.property.file` system property as you start the free-text search engine. Upon starting, the free-text search engine loads and uses the contents of the given file for substitution in file `solrserver-config.xml`. The free-text search engine logs the substitution activity in the free-text search engine log.

Plugin ExternalConfigurationProviderPlugin

The Substitutor process queries the `ExternalConfigurationProviderPlugin` plugin, using the placeholder key and namespace, to determine the replacement value for the placeholder. In the base configuration, Guidewire provides a default implementation class for this plugin. It is also possible to create your own implementation class for the `ExternalConfigurationProviderPlugin` plugin.

Enabling the ExternalConfigurationProviderPlugin plugin

In the base configuration, Guidewire disables the `ExternalConfigurationProviderPlugin` plugin by default. To use the plugin functionality, you must first enable the plugin in the Studio plugin editor. After you open the plugin in the Studio plugin editor, uncheck the **Disabled** box.

ExternalConfigurationProviderPlugin - default implementation

In the base configuration, Guidewire provides a default implementation class for plugin `ExternalConfigurationProviderPlugin`:

```
FileBasedConfigurationProviderWrapper
```

The default class implements the `ExternalConfigurationProviderPlugin` interface.

Whatever the plugin implementation class, the plugin implementation class must contain the following public method:

```
lookupValue(String namespace, String key)
```

In this method:

- `namepace` is a hard-coded string that uniquely identifies a given file group.
- *`key`* is the identifier for the configuration placeholder .

The Substitutor process calls the `lookupValue` method every time it encounters a placeholder in a supported configuration file. The plugin method call then returns a substitution value for placeholder, which the Substitutor uses to replace the entire placeholder in the configuration file.

Placeholder examples

The following examples illustrate how to work with the `lookupValue` method:

- The Substitutor replaces placeholder `${url}` in file `config.xml` with the value returned by method `lookupValue("config", "url")`.
- The Substitutor replaces placeholder `${url}` in file `myPlugin.gwp` with the value returned by method `lookupValue("plugin", "url")`.

Lookup failure

If there is no default value and the plugin does not return a value:

- PolicyCenter leaves the placeholder strings unchanged.
- PolicyCenter places warning messages in the application log.

If a placeholder remains in a file used by the application server, it can cause the application server to fail, if not excluded from consideration due to an `env` or `server` configuration.

ExternalConfigurationProviderPlugin - custom implementation

Any implementation class that you create for plugin `ExternalConfigurationProviderPlugin` must implement the `ExternalConfigurationProviderPlugin` interface. In addition, your implementation class must contain a public `lookupValue(namespace, key)` method that behaves in the same way as the default implementation of this method in implementation class `FileBasedConfigurationProviderWrapper`.

It is up to you to determine where and how PolicyCenter takes the value for substitution. If you do not provide your own plugin implementation:

- PolicyCenter ignores all placeholders without default values.
- PolicyCenter replaces all placeholders with default values with the given default value.

Custom plugin considerations

You must name your plugin implementation `ExternalConfigurationProviderPlugin.gwp` exactly, as PolicyCenter loads the plugin by name.

Guidewire recommends that you be mindful of performance considerations in your implementation of this plugin. It is a good idea to use caching so as to avoid expensive I/O reads as PolicyCenter accesses the plugin for every substitution that it finds in all supported files.

As the `ExternalConfigurationProviderPlugin` plugin is a special type of plugin (an early-load plugin), it needs to be available even before PolicyCenter parses the first configuration file (`config.xml`) for the first time. Thus, your implementation of this plugin must follow the following implementation rules:

- File `ExternalConfigurationProviderPlugin.gwp` cannot itself contain placeholders.
- Plugin `ExternalConfigurationProviderPlugin` must be either a Gosu- or Java-based plugin. It explicitly cannot be an OSGI plugin.

Your custom plugin implementation must also adhere to the following restrictions:

- The implementation class cannot use Guidewire entities.
- The implementation class cannot access the PolicyCenter database.
- The implementation class cannot use server lifecycle-dependent code.
- The implementation class cannot use server lifecycle-dependent APIs.

Accessing ExternalConfigurationProviderPlugin from code

Guidewire provides the following public API that provides access to the `ExternalConfigurationProviderPlugin` plugin:

```
gw.external.configuration.SubstitutionProperties
```

This API contains a single `lookupValue` method that internally delegates the API method call to the identical method in the `ExternalConfigurationProviderPlugin` plugin implementation. The following example illustrates how to work with this API.

```
uses gw.external.configuraton.SubstitutionProperties

var value = SubstitutionProperties.lookupValue(namespace, key)
```

Verifying the external configuration

Guidewire provides a `gwb` build command that you can use to verify your externalized server configuration. Use the following syntax at a command prompt to execute the command:

```
gwb verifyExtConfig
```

This command generates a report that includes the following types of information:

- The number of files affected
- The number of substitutions involved
- The status of plugin `ExternalConfigurationProviderPlugin` if disabled

Example

Running the verifyExtConfig command generates a report similar to the following text, if there are no issues with the externalized server configuration:

```
Verification results:

14 key substitutions would occur in 3 files.
```

Path to the substitution file

Notice in the `gwb verifyExtConfig` command that you do not specify a path to the substitution file. Instead, you set the path to the external substitution file as you start the application server, for example:

```
gwb runServer -Dgw.config.external.property.file="C:\Guidewire\10.0\cc10_1853\modules
\configuration\etc\ext-properties.properties"
```

Processes that support configuration substitution

Variable substitution touches every PolicyCenter process that involves parsing configuration files that support externalization. It is possible to divide these processes into the areas listed in the following table. For each group of processes, different aspects of substitution apply.

PolicyCenter process	More information
Stand-alone application server	"Stand-alone PolicyCenter servers" on page 78
Clustered application servers	"Clustered PolicyCenter servers" on page 79
System tools	"PolicyCenter system tools" on page 79
Build tasks	"PolicyCenter gwb tasks" on page 79

Stand-alone PolicyCenter servers

If you enable external server configuration on a stand-alone PolicyCenter server, PolicyCenter performs a series of steps in working with placeholders:

1. Initially, at server start, PolicyCenter reads the affected configuration files, once, during one of the various phases of server start-up.

2. The PolicyCenter Substitutor process then performs an in-place substitution step on all affected configuration files.
3. Finally, PolicyCenter parses the contents of the modified configuration files and stores the results in memory.

Clustered PolicyCenter servers

During the start of the first server node in the PolicyCenter cluster, PolicyCenter creates a digest of all the configuration files and their parameters that are present on the start-up node. PolicyCenter then stores this configuration information in the database. This information becomes the official configuration information for the entire cluster.

In the usual course of events, as each subsequent PolicyCenter node starts, PolicyCenter performs a configuration verification for that node. If there is a difference between the configuration for that node and the configuration information stored in the database, PolicyCenter decides how to handle the difference in the two configurations:

- If the new configuration is non-compatible with the existing cluster configuration, PolicyCenter requires a full database upgrade.
- If the difference between the two configuration is within defined parameters, PolicyCenter requires a rolling upgrade only.

It is possible to have two seemingly identical configuration files that differ in their effect by including or excluding parts of each configuration through the use of `env` or `serverid` settings. PolicyCenter creates different digests from these configuration files. Be aware that PolicyCenter considers these digests to be different from each other and potentially non-compatible.

PolicyCenter system tools

After you run the `system_tool -verifyconfig` command against a running PolicyCenter server, the command uses the remote `ExternalConfigurationProviderPlugin` plugin implementation to determine the substitution values for the local configuration files, if any. The command then compares the digest extracted from the local WAR/EAR file configuration files with the digest of the remote cluster configuration. Finally, the command prints the result of this comparison to the command prompt in which you ran the `system_tool` command.

PolicyCenter gwb tasks

The following `gwb` build tasks all parse the configuration files that support substitution:

- `-dropDb`
- `-runServer`
- `-diffDisplayKeys`
- `-exportLocalization`
- `-importLocalization`
- `-genFromWsc`
- `-genJson`
- `-genWsiLocal`
- `-genDataDictionary`
- `-genEntityModelXml`
- `-genDataMapping`
- `-gemImportAdminDataXsd`
- `-genPcfMapping`
- `-verifyExtConfig`
- `-verifyResources`

Build command `-verifyExtConfig` checks that the `ExternalConfigurationProviderPlugin` plugin provides all necessary values for configuration substitution in the affected files. The command output reports any issues that it finds.

Logging substitution activity

PolicyCenter outputs all substitution-related messages to the application log using the `Server.PropertySubstitutor` logger. The logging output also includes information on the initial loading of the `ExternalConfigurationProviderPlugin` plugin. The logging messages are verbose and sufficient to track all steps and problems of any substitution operation.

chapter 6

PolicyCenter Server administration

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

How to start Guidewire PolicyCenter

How you start PolicyCenter depends on the application server type.

Application Server	More information...
QuickStart (Jetty)	• "Start PolicyCenter on QuickStart (Jetty)" on page 81 • "Start the application server from Guidewire Studio for PolicyCenter" on page 82
JBoss	*Installation Guide*
Tomcat	*Installation Guide*
WebLogic	*Installation Guide*
WebShpere	*Installation Guide*

Start PolicyCenter on QuickStart (Jetty)

About this task

Guidewire recommends the following sequence in starting PolicyCenter on the QuickStart server.

Procedure

1. Open a command prompt and navigate to the root of PolicyCenter application directory.
2. Run the following command to compile the needed application resources and move them to the correct location in the application server:

   ```
   gwb compile
   ```

3. Run the following command to start the application server.

   ```
   gwb runServer -x compile
   ```

   There is a dependency between the `runServer` command and the `compile` command. Guidewire recommends that you always use the `-x compile` option with the `runServer` command to remove that dependency after you initially run the `compile` command. Otherwise, PolicyCenter must first verify what resources, if any, need to be recompiled, then perform an incremental recompile of those resources before starting the server.

See also

- “JVM options specific to the runServer build command” on page 66
- *Installation Guide*

Start the application server from Guidewire Studio for PolicyCenter

It is possible to start the PolicyCenter development server (Jetty) from Guidewire Studio™ for PolicyCenter.

Procedure

1. Navigate to the following location in Guidewire Studio™ for PolicyCenter, using the menu bar at the top of the screen:

   Run→Run...

   Studio opens a **Run** drop-down with a list of server-related options.
2. Select **Edit Configurations..** from the list.

   Studio opens the **Run - Server** dialog.
3. Select **Server** in the left-hand navigation pane and verify the configuration options set for the server.

   It is possible to modify the base configuration server settings by adding additional VM options. If you do so, use the following format (using server ID an example):

```
-Dgw.pc.serverid=testServer
```

4. Click **Run**.

   Studio opens a pane at the bottom of the screen to display the server log. This area also has controls (icons) for stopping and starting the server.

How to stop Guidewire PolicyCenter

Before you stop Guidewire PolicyCenter, you must stop all work queues. Distributed workers run on daemon threads. As the JVM (Java Virtual Machine) exits, it destroys these threads. This can cause issues if the JVM destroys a thread while that thread is processing a work item. For example, suppose that a work queue calls a plugin that makes a blocking call to an external system or otherwise take a long time to return. In that case, if you do not shut down the work queue threads correctly, it is possible to end up with inconsistent data.

Stop Guidewire PolicyCenter

Procedure

1. Login into Guidewire PolicyCenter as a user with administrative privileges.
2. Within PolicyCenter, press `ALT+SHIFT+T` to display the **Server Tools** screens.
3. Navigate to **Batch Process Info** screen:
   a. For any process that has a **Last Run Status** that indicates that the process is running, click the **Stop** button in the **Action** column.
   b. For any process that has a **Next Scheduled Run** time that is before the time that you intend to stop PolicyCenter, click **Stop** in the **Schedule** column.

      All processes must have a **Status** of **Completed** before you stop the PolicyCenter server.
4. Stop Guidewire PolicyCenter:
   - To stop PolicyCenter in a production environment, stop the server on which it is running.
   - To stop PolicyCenter in a development environment, run the following command from the PolicyCenter installation directory:

```
gwb stopServer
```

Server startup tests

The PolicyCenter server performs a series of tests during start up. For some of these tests, a test failure prevents the server from starting. For other tests, a test failure is simply a warnings and PolicyCenter permits the server to start. In many cases, these checks warn about potential problems with the archive and domain graphs, but which might not be an issue depending on business logic.

See also

- “The Domain Graph Info screen” on page 364
- “The Consistency Checks screen” on page 364
- *Configuration Guide*
- *Product Model Guide*

Server modes

Server mode determines what functionality is available at various server run levels. All PolicyCenter server types, except for QuickStart, can run in any of the following server modes:

- Development
- Test
- Production

PolicyCenter starts in production mode on all supported servers by default, except for 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.

See also

- “Server test mode” on page 84
- “Server run levels” on page 84
- “Setting the server mode” on page 84
- “Set the QuickStart run level at server start” on page 86
- *Installation Guide*

Important server mode caveats

The following caveats are important in working with the Guidewire development and production databases:

- It is not permissible to start a server in development mode using a production mode database.
- It is not permissible to start a server in production mode using a development mode database. Starting the server in production mode expressly does not upgrade the development mode database to production mode.

Server modes as a safety feature

Guidewire provides the server modes as a safety precaution so that it is not possible to use development tools on a production server. Some system functions are useful for development, but are not appropriate, or, are dangerous if used in a production environment. Thus:

- In development and test modes, it is possible to access both the **Server Tools** and **Internal Tools** screens.
- In production mode, it is possible to access the **Server Tools** screens only.

For more information on these tools, see “Server tools” on page 353 and “Internal tools” on page 417.

Setting or resetting the system time is also not safe to do in a production environment. The use of the `ITestingClock` plugin 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, it is possible for the use of this plugin in a production environment to cause disastrous results. Therefore, you can only use this plugin if the server is in development or test mode.

It is not possible to deploy certain types of product model changes to a server set to production mode. Guidewire provides these locks to protect data integrity. Any time that is important to upgrade (or preserve) the database, place the server into production or test mode to minimize the risk of data corruption.

See also

- *Product Model Guide*

Server test mode

Other than the exceptions listed in the following table, server test mode is identical to server production mode.

Functionality	Test mode	Production mode
System clock	You can adjust the testing system clock by using the `setCurrentTime` method on the `ITestingClock` plugin. **See also** • "The Testing System Clock screen" on page 418 • *Integration Guide*	Not available
Server Tools screens	Available	Available
Internal Tools screens	Available	Not available
Console and log file	PolicyCenter prints a message to the console during startup indicating that the server is running in test mode.	PolicyCenter prints a message to the console during startup indicating that the server is running in production mode.
PolicyCenter title bar	The title bar for a browser connected to PolicyCenter indicates that PolicyCenter is in test mode.	Not applicable

Setting the server mode

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

To set the server mode at server start, use the following system parameter:

```
-DserverMode={dev|prod|test}
```

To change the mode of a running server, restart the server and set the `-DserverMode` parameter to `dev`, `test` or `prod`.

PolicyCenter ignores this parameter on the QuickStart server.

Determining the server mode

You can determine the server mode through one of the following methods:

- By reading the console log as you start PolicyCenter
- By checking the title bar of the browser in which PolicyCenter is running

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

Server run levels

Guidewire PolicyCenter supports the concept of a server *run level*. A run level provides the ability to start the PolicyCenter server with a specific level of functionality. For example, during a full application and database upgrade, Guidewire recommends that you start PolicyCenter with limited functionality, in MAINTENANCE mode.

Guidewire specifies the server run levels in several different ways:

- A numeric run level, ranging from 0 to 5, used with the development QuickStart server.
- A server run level name that corresponds to the numeric run levels
- A system run level name used with the administrative system tools to set the server to one of the server run levels

The following table lists the correspondence between the various ways of describing the server run levels.

QuickStart	Server run level	System run level
0	NONE	—
1	GUIDEWIRE_STARTUP	—
2	SHUTDOWN	—
3	NODAEMONS	maintenance
4	DAEMONS	daemons
5	MULTIUSER	multiuser

The following list describes each type of server run level in more detail.

Type	Description
QuickStart run level	Set at QuickStart server start using the following command, with *n* being a specific run level number: `gwb runServer --run-level n` See "Set the QuickStart run level at server start" on page 86.
Server run level	Shown in the server log. The server starts at level 0 and proceeds to move through each server run level in the sequence until arriving at the requested run level.
System run level	Set through command prompt `system_tools` options, for example: `system_tools -maintenance` See "system_tools command" on page 429 for details.

Server run levels are independent of the server mode. The combination of mode and run level determines the availability of functionality, such as the user interface and web services.

See also

- "Server modes" on page 83
- "Server modes and server run levels" on page 85
- "Place the server in maintenance mode" on page 89
- *Installation Guide*

Server modes and server run levels

The following table shows which functionality is available for the possible combinations of server modes and server run levels. For a description of the numeric values used with the development QuickStart server, see "Server run levels" on page 84.

Quick-Start	Server run level	Production mode	Development mode
0	NONE	• Nothing available.	• Nothing available.
1	GUIDEWIRE_STARTUP	• User interface not available. • Web services not available. • Database not available.	• User interface not available. • Web services not available. • Database not available.

Quick-Start	Server run level	Production mode	Development mode
2	`SHUTDOWN`	• User interface not available. • Web services not available. • Database not available.	• User interface not available. • Web services not available. • Database not available.
3	`NODAEMONS`	• 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.
4	`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 PolicyCenter, such as those used to dispatch messages.	• User interface available. All logins allowed. • Web services available. • Work queues (including workflow) available. • Workflow Stat Manager available. • Scheduler available. • Daemons started by PolicyCenter, such as those used to dispatch messages.
5	`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. • Server Tools available to all users if `EnableInternalDebugTools` is set to `true` in `config.xml` • **Internal Tools** available. • Web services available.

Setting the QuickStart server run level

It is possible to set the run level of a server to one of the following system run levels:

- `daemons`
- `maintenance`
- `multiuser`

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

See also

- "Server run levels" on page 84
- "Server modes and server run levels" on page 85
- "system_tools command" on page 429
- *Integration Guide*

Set the QuickStart run level at server start

Procedure

1. Open a command prompt and navigate to the PolicyCenter installation directory.
2. Run the following command:

```
gwb runServer -run-level n
```

The value of *n* is the numeric value of the run level as defined in “Server run levels” on page 84.

Set the server run level through system tools

Procedure

1. Ensure that the PolicyCenter server is running.
2. Open a command prompt and navigate to the following location in your PolicyCenter installation directory:

```
admin/bin
```

3. Enter one of the following commands:

```
system_tools -user user -password password -daemons
system_tools -user user -password password -maintenance
system_tools -user user -password password -multiuser
```

You must supply the username (*user*) and password (*password*) for a user with administrative privileges on the PolicyCenter server. The run level is a value as defined in “Server run levels” on page 84.

Set the server run level through web services

Procedure

1. Ensure that the PolicyCenter server is running.
2. Call the following method on the `SystemToolsAPI` web service:

```
SystemToolsAPI.setRunLevel
```

Next steps

If you run PolicyCenter in a clustered environment, you cannot place all the computers in a particular run level with a single method call. Instead, you must call the method individually on each cluster member.

Determining the server run level

You can determine the server run level through any of the following methods.

Type	More information
System tools	“Use system tools to determine the server run level” on page 88
Web services	“Use web services to determine the server run level” on page 88
Server ping	“Using the PolicyCenter ping utility” on page 183

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

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

See also

- "Server run levels" on page 84

Use system tools to determine the server run level

Procedure

1. Ensure that the PolicyCenter server is running.
2. Open a command prompt and navigate to the following location in the PolicyCenter installation directory:

```
admin/bin
```

3. Enter the following command:

```
system_tools -user user -password password -ping
```

You must supply the username (*`user`*) and password (*`password`*) for a user with administrative privileges on the PolicyCenter server.

Use web services to determine the server run level

Procedure

1. Ensure that the PolicyCenter server is running.
2. Call the following method on the `SystemToolsAPI` web service:

```
SystemToolsAPI.getRunLevel
```

See also

- *Integration Guide*

Server maintenance mode

Periodically, you need to perform maintenance on PolicyCenter, for example, importing new security roles. To prevent users from logging into PolicyCenter during these activities, place PolicyCenter into the maintenance run level.

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 prompt tools for any daemons with a minimum run level equal or lower than `NODAEMONS`. Restricting the run level permits integration processes to proceed without interference from non-administrator users.

See also

- "Server run levels" on page 84
- "Set the QuickStart run level at server start" on page 86
- "Set the server run level through system tools" on page 87
- "Set the server run level through web services" on page 87

Place the server in maintenance mode

Procedure

1. Open a command prompt and navigate to the following PolicyCenter installation directory:

```
admin/bin
```

2. Run the following command:

```
system_tools -password password -user user -maintenance
```

You must supply the username (*user*) and password (*password*) for a user with administrative privileges on the PolicyCenter server.

Monitoring server performance in PolicyCenter

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 also

- “Monitoring server status with WebSphere” on page 89
- “Monitoring cluster health” on page 181
- Documentation specific to the application server
- *Installation Guide*

Monitoring server status with WebSphere

You can monitor the status of the PolicyCenter server from the WebSphere console. To view the server’s status, select the PolicyCenter cluster member from the main console. WebSphere displays the **Show Status** option if the server is running. WebSphere also generates and displays system logs. Also, you can start an **Export for Backup** from the WebSphere console. Guidewire recommends that you back up the server before performing any major system maintenance.

Monitoring and managing event messages

PolicyCenter generates a large number of events. In a PolicyCenter installation, it is often helpful, or even necessary, for PolicyCenter to notify other applications of these events. 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.

Guidewire recommends that you monitor message traffic to verify that the integration is running smoothly.

See also

- For more information about messages, including how to create message destination objects, see the *Integration Guide*.

About message failure

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 a message error, do one of the following:

- Manually retry to send the message
- Skip the message

You can retry or skip messages:

- Through the **Message Queues** screen available to application administrators
- Through `messaging_tools` command prompt options

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

See also

- "The Message Queues screen" on page 91
- "messaging_tools command" on page 427

The Messages screen

Use the PolicyCenter **Messages** screen to review information about messages. An administrator can access the **Messages** screen in PolicyCenter by using the following navigation path:

Administration→Monitoring→Messages

You can search and filter message by any of the following:

- **Destination**
- **Transaction Type**
- **Transaction Number**
- **Product**
- **Policy Number**
- **Account Number**
- **Message Status** – For each status, the corresponding status is:
  - **Failed messages** – `MessageStatus.ERROR_STATES`
  - **Messages needing retry** – `MessageStatus.RETRYABLE_STATES`
  - **Unfinished messages** – Messages that do not fall into `ERROR_STATES` or `RETRYABLE_STATES`

The messages in **Results** have the following information:

- **Transaction Number**
- **Transaction Type**
- **Policy Number**
- **Product**
- **Primary Named Insured**
- **Message Status**
- **Error Description**

See also

- *Integration Guide*

The Message Queues screen

PolicyCenter lists each message destination in the **Message Queues** screen. You access this screen from the PolicyCenter **Administration** screen by first choosing **Monitoring**. The first screen of **Message Queues** contains cumulative information about message destinations.

From this screen, you can select a message destination and perform the following actions:

- **Suspend** – Click to suspend the operation of the selected message destination.
- **Resume** – Click to resume the operation of the selected message destination if that destination is in a state of suspension.
- **Restart** – Click to restart the operation of a selected message destination.

You can also restart the messaging engine by clicking **Restart Messaging Engine**.

If a message destination is running correctly, you do not see any accumulation of information in the columns on this screen. If there is a problem and messages begin to accumulate, you can drill down into a message destination by clicking the destination name. This action opens a **Destination** screen. From the **Destination** screen, you can see additional detail about any issues with 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 issue.

The Destination screen

The **Destination** screen lists all failed or in-process messages for an account for all destinations. You can search for a particular account and then open the account's detail view. From this screen, you can select one or more objects and instruct PolicyCenter to skip, retry, or re-synchronize any message that is still in process or in a failed state.

The Detailed Statistics table column headers have the following meanings.

Column header	Meaning
Safe Ordering Object Name	PolicyCenter groups messages for each messaging destination based on their associated primary object. (PolicyCenter processes messages associated with objects other than the primary object as non-safe-ordered messages.)
Send Time	Time the PolicyCenter sent the message.
Failed	A message can fail for several reasons, for example: • The message destination did not process the message successfully due to a processing error. • The message destination returns a negative acknowledge (`nack`) indicating that the message delivery failed. • The message was part of a series of messages, one or more of which failed.
Retryable Error	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.
Unsent	The message has not been sent, for example: • The message is waiting on a prior message. • The destination is not processing messages as the destination is in a state of suspension. • The destination is falling behind in processing messages.
Error Message	Error message returned if a message fails.

It is possible to filter the messages that show in the table by selecting a filtering characteristic from the filtering drop-down list.

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, suppose that a messaging server starts writing message 1 to the queue, and then starts writing message 2 to the same queue. It is possible that the server completes and commits message 2 while still writing message 1. 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 message 2. You are most likely to encounter this situation in a clustered PolicyCenter environment.

To address this situation, PolicyCenter provides the `IncrementalReaderSafetyMarginMillis` parameter in file `config.xml`. This parameter determines how long after detecting a skipped message that PolicyCenter attempts to read messages again. This waiting period gives PolicyCenter a chance to commit the skipped message. If it is not possible to commit the message before the expiration of the waiting period:

- PolicyCenter assumes the message is lost and that it is not possible to commit the message.
- PolicyCenter skips the message permanently, thereafter.

For example, in the previous scenario, PolicyCenter waits 10 seconds (the default parameter value) before attempting to read messages again, beginning with the skipped message 1. If message 1 has still not been committed at that time, PolicyCenter skips it permanently.

Set the `IncrementalReaderSafetyMarginMillis` parameter sufficiently long so that the server can commit the messages, but without prematurely marking messages as permanently skipped. As the server does not read any other messages during this waiting period, do not set `IncrementalReaderSafetyMarginMillis` so long as to delay the delivery of messages.

You can also set the following configuration parameters in `config.xml` to configure the messages reading environment:

- `IncrementalReaderPollIntervalMillis`
- `IncrementalReaderChunkSize`

Access the messaging editor

About this task

You create and configure message environments and destinations in the file named `messaging-config.xml`.

Procedure

1. In the PolicyCenter Studio **Project** window, navigate to **configuration→config→Messaging**.
2. Double-click on the file `messaging-config.xml` to open it in the Studio **Messaging** editor.

See also

- *Configuration Guide*

Purge completed messages

About this task

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. As the number of messages in PolicyCenter can grow to be large, Guidewire recommends that you purge completed messages on a regular basis.

Procedure

1. Ensure that the Guidewire PolicyCenter server is running.

2. Open a command prompt in the following location in the PolicyCenter installation directory:

```
admin/bin
```

3. Enter the following `messaging_tools` command, replacing the variables as needed:

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

For example, the following command purges all completed messages received prior to `02/06/06`.

```
messaging_tools -user su -password gw -purge 02/06/17
```

You must supply the username (*`user`*) and password (*`password`*) for a user with administrative privileges.

Session timeout

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

You configure the timeout value for a session by setting the `SessionTimeoutSecs` parameter in `config.xml`. This value sets the session expiration timeout globally for all PolicyCenter browser sessions.

Typically, the 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 the timeout value is not set 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`.

See also

- *Configuration Guide*

User session replication

Do not attempt to replicate sessions across PolicyCenter cluster members. PolicyCenter does not implement or support user session replication for a number of reasons, including the following:

- 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 large amounts of information. Guidewire estimates that this information amounts to 1 MB of data on average for a 32-bit server and close to 2 MB for a

64-bit server. Session replication would create significant cross-member communication that is detrimental to performance.

- PolicyCenter commits changes to the database on almost all transactions. Notable 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.

Instead, Guidewire recommends that you implement a PolicyCenter cluster consisting of multiple PolicyCenter application instances with failover and a load balancing solution. The load balancer must implement *session affinity*, meaning that it must route connections from the same user session to the same PolicyCenter server.

See also

- "Planning a PolicyCenter cluster" on page 177
- *Installation Guide*

Cache management

Objects do not remain forever present or valid in the PolicyCenter database cache. Guidewire provides several caching mechanisms to verify that cache entries are still relevant. They are:

- Stale timeout
- Eviction timeout
- Cluster member object tracking

Stale timeout

A stale timeout mechanism ensures that the server instance does not use old object entries excessively. 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 instance calculates the duration since the object was last read from the database. If that duration exceeds the stale time, the server instance 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`.

Eviction timeout

A PolicyCenter evict timeout mechanism removes old objects from the cache. For example, an object has an evict time of 15 minutes and a stale time of 30 minutes. If the server uses the object a single time 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`. In the base configuration Guidewire sets the value of `GlobalCacheReapingTimeMinutes` to 15 minutes. The minimum value for this parameter is 1 minute. The effective maximum value for parameter `GlobalCacheReapingTimeMinutes` is the lesser of its set value or the value of `GlobalCacheStaleTimeMinutes` parameter.

Cluster member object tracking

Upon receipt of an inter-cluster message indicating that a cluster member changed an object value, the other cluster members mark the corresponding entry in the cache as obsolete. The object value then becomes available for reuse.

See also

- "Guidewire PolicyCenter cluster installations" on page 162
- "The Cache Info screens" on page 404
- *Configuration Guide*

Caching and stickiness

PolicyCenter is fully leveraging the caching mechanism if a user returns to the same server instance across different HTTP requests. In a clustered environment, the load balancer must direct requests to the same PolicyCenter server 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 server manages its own cache. It is possible for the same object to live in the cache of two or more servers at the same time. Some object sets, such as users, likely live in the global cache of all servers in a cluster.

Concurrent data change prevention

It is possible for different users, either on the same PolicyCenter server or on a different server, to attempt to change data objects concurrently. To prevent data corruption, Guidewire implements a data object versioning mechanism. During the update to 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 two different user changed the object concurrently. In such case, PolicyCenter rejects the change and the cache mechanism throws a concurrent data 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 data 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, the locking mechanism blocks any other user of batch process from making a change. Pessimistic locking systems generally become impractical. Therefore, PolicyCenter uses the optimistic locking mechanism.

Caching and clustering

For information on how Guidewire clusters handle caching, see “Guidewire PolicyCenter cluster installations” on page 162.

Cache behavior and performance

Guidewire generally distinguishes two types of caches:

- Server cache – A cache that is purely local to the PolicyCenter server
- Database cache – A cache used by a database to hold data retrieved from storage

Proper database caching behavior is critical to performance.

The server cache is purely local to each individual PolicyCenter server. Therefore, one server instance might contain information on a specific object while another server might not contain that information.

For example, if a PolicyCenter user works on a policy, PolicyCenter loads corresponding objects on the server to which the user connects. If another user must approve the action of the first user, the approver user might interact with another PolicyCenter server. In that case, the second server likely does not have the corresponding information in cache. Therefore, the approver might experience slower performance as the server must populate the cache.

Cache content is lost every time that you stop a PolicyCenter server. Therefore, after you start a PolicyCenter server, expect lower performance during a ramp-up phase.

Batch processes leverage the cache mechanism. Batch jobs can work on many objects. Therefore, batch jobs can use the cache extensively. This can have the adverse effect of prematurely evicting objects from the cache, thereby forcing additional cache loads. If you run many intensive batch processes, Guidewire recommends that you have a dedicated server with the `batch` role with no online traffic directed to it.

See also

- “Server cache tuning parameters” on page 97

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 onto a server that has the `batch` server role. 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. For this reason, Guidewire recommends that you have separate servers for handling batch processing and user interface transactions.

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

See also

- “Detect cache thrashing” on page 96.

Detect cache thrashing

Procedure

1. Log into Guidewire PolicyCenter as a user with administrative privileges.
2. Press `ALT+SHIFT+T` to open the **Server Tools** screens.
3. Navigate to the **Cache Info** screen.
4. Use the information in the **Cache Info** screen to analyze the number of evictions.
5. Click **Clear Global Cache** to clear the cache.
6. Reproduce the operation that you suspect caused the cache thrashing.
7. Reanalyze the information on the **Cache Info** screen.

Next steps

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

See also

- “Cache thrashing” on page 96.

Cache impact on memory utilization

The maximum size of the cache is dependent on cache parameters. The 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, it is difficult to reliably estimate the maximum size of a cache. 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 also

- “Server cache tuning parameters” on page 97.
- “Server memory management” on page 99

Server cache tuning parameters

File `config.xml` 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. PolicyCenter uses this specialized cache for exchange rates only.
`GlobalCacheActiveTimeMinutes`	Time, in minutes, that PolicyCenter considers cached objects active. You can think of this as a period in which PolicyCenter is heavily using an item, for example, how long a user stays on a screen. The cache mechanism gives higher priority to preserving these higher-use objects. Set `GlobalCacheActiveTimeMinutes` to a value less than `GlobalCacheReapingTimeMinutes`.
`GlobalCacheDetailedStats`	Boolean value that specifies whether to collect detailed statistics for the global cache. Detailed statistics are data that PolicyCenter collects to explain why the caching mechanism evicted items from the cache. PolicyCenter collects basic statistics, such as the miss ratio, regardless of the value of `GlobalCacheDetailedStats`. Disabling collection of detailed cache statistics can sometimes improve performance. Guidewire sets the value of `GlobalCacheDetailedStats` 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** screen does not include the **Evict Information** and **Type of Cache Misses** graphs. At runtime, use the **Management Beans** screen to enable the collection of detailed statistics for the global cache.
`GlobalCacheReapingTimeMinutes`	Time, in minutes, since the 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. PolicyCenter uses this specialized cache for group-related data only.
`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** screen to modify this value.

Parameter	Description
`GlobalCacheSizePercent`	Maximum amount of heap space used to store cached entities, expressed as a percentage of the maximum heap size.
`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** screen to modify this value.
`GlobalCacheStatsWindowMinutes`	This parameter denotes a period of time, in minutes, that PolicyCenter uses for two purposes: • The period of time to preserve the reason that PolicyCenter evicted an object, after the event occurred. If a cache miss occurs, PolicyCenter reports the reason on the **Cache Info** screen. • The period of time to display statistics on the chart on the **Cache Info** screen.
`ScriptParametersRefreshIntervalSecs`	The number of seconds between refreshes of the script parameters cache. PolicyCenter uses this specialized cache for script parameters only.
`ZoneCacheRefreshIntervalSecs`	The number of seconds between refreshes of the zones cache. PolicyCenter uses this specialized cache for zones only.

See also

- "Special caches for rarely changing objects" on page 99
- "The Cache Info screens" on page 404
- "The Management Beans screen" on page 390
- *Configuration Guide*

Understanding cache metrics

Cache hit ratio metric information intrinsically depends 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.

Also, it is also for the data be appear skewed to the low side if a server started recently, or if the server has not had much user load. For example, if you recently started the server, and users have only visited a few screens, the hit rate is very low because PolicyCenter encountered only a few cache hits. As users visit more pages, the hit rate increases.

Thus, there are no good default cache hit ratios. Experimentation combined with performance measurements constitutes the only approach to identifying appropriate cache sizes.

See also

- For information on how to view cache performance, see "The Cache Info screens" on page 404.

Special caches for rarely changing objects

PolicyCenter includes specific individual caches for the following rarely changing objects:

- Exchange rates
- Groups
- Script parameters
- Zones

These specialized caches periodically refresh the entire set of the rarely changing object. This prevents the 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 through a configuration parameter in file `config.xml`.

The following table lists the objects with specialized caches, along with the configuration parameter that controls the refresh rate of each specialized cache.

Cache objects	Cache refresh rate set by…
`ExchangeRate`	`ExchangeRatesCacheRefreshIntervalSecs`
`Group`	`GroupCacheRefreshIntervalSecs`
`ScriptParameter`	`ScriptParametersRefreshIntervalSecs`
`Zone`	`ZoneCacheRefreshIntervalSecs`

See also

- "Server cache tuning parameters" on page 97.
- *Configuration Guide*

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. In general, computer science calls this term garbage collection. Garbage collection can have a significant impact on server performance.

This topic describes Java platform garbage collection analysis.

Memory usage logging

If configured in file `log4j2.xml`, Guidewire logs contain memory usage messages that provides information on the use of memory by the JVM. To enable memory usage logging, you must set parameter `MemoryUsageMonitorIntervalMins` in `config.xml` to a value other than the default of 0.

A memory usage message looks similar to the following:

```
serverName 2016-04-09 16:44:14,423 INFO Memory usage: 80.250 MB used, 173.811 MB free, 254.062 MB total,
      2048.000 MB max
```

The following list describes the different types of information that you see in a memory logging message.

Memory usage	Meaning
`used`	Amount of memory allocated to objects. This includes memory for the following memory types: • Memory used by active objects still in use • Memory used by stale objects (on which the JVM eventually performs garbage collection)
`free`	Amount of un-allocated memory
`total`	Amount of memory that the JVM process reserves from the operating system.

Memory usage	Meaning
`max`	Amount of maximum total memory that PolicyCenter allows the JVM to use.

It is common for a 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. If the server needs more memory, it triggers garbage collection to free up the memory used by stale objects.

Memory usage messages and garbage collection

It is not possible to configure the content of the logging messages to provide enough detail to indicate or warn of memory issues. The only way to obtain a more accurate picture of memory usage is through running the garbage collector.

Guidewire does not support running the garbage collector merely for the sake of more detailed logging. However, to obtain more detailed information on PolicyCenter memory usage in the current configuration, enable garbage collector logging.

You only need to worry about memory issues if the server throws an `OutOfMemoryError` exception. If that happens, Guidewire recommends that you configure the garbage collector to print out detailed memory information.

See also

- “Enabling garbage collection” on page 100
- “Understanding possible memory leaks” on page 101

Enabling garbage collection

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

Verify that the performance analysis tool you choose supports the version of the JVM that you use for PolicyCenter. To determine the supported JVM versions, visit the Guidewire Community and search for knowledge article 1005, *Supported Software Components*.

See also

- “Enabling verbose garbage collection for IBM JVM” on page 100
- “Enabling verbose garbage collection for Oracle Java Hotspot JVM” on page 101

Enabling verbose garbage collection for IBM JVM

To enable verbose garbage collection for IBM Java Virtual Machines, add 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. In general, there is no need for additional levels of logging.

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.

The IBM Support Assistant Workbench

IBM provides the IBM Support Assistant Workbench. It is possible install multiple plugin tools 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 work more 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 the following web site for more information about the IBM Support Assistant:

```
http://www.ibm.com/software/support/isa/
```

Enabling verbose garbage collection for Oracle Java Hotspot JVM

To enable verbose garbage collection on an Oracle Java Hotspot JVM, add 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.

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.

For information on these utility tools, refer to the respective company web sites.

Refer to the following web sites for more information:

- **HPjmeter** – `http://www.javaperformancetuning.com/tools/hpjmeter/index.shtml`
- **GCViewer** – `http://www.tagtraum.com/gcviewer.html`

Understanding possible memory leaks

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. Computer science commonly describes this condition as a *memory leak*. To diagnose the problem, it is necessary to collect a dump of all used objects, called a *heap dump*, to identify all objects in the heap. Developers familiar with PolicyCenter can then analyze the heap dump. Such analysis helps identify excessive memory usage, identify its root cause and possibly find a change that will avoid such issues.

The investigation of memory leaks differs slightly depending on the JVM platform.

Common approaches for heap dump generation

In general, you can use the following options to generate a heap dump:

1. Set specific flags to force the following behaviors:
   - Heap dump generation if the heap is full and an out-of-memory condition occurs
   - Heap dump generation if an application `CTRL-BREAK` or SIGQUIT occurs

   Combine these options with options instructing the JVM to generate the heap dump at a specific directory location.
2. Use tools to connect to a running JVM. Such tools provide the option to trigger a heap dump.

Generating heap dumps with IBM JDK

In general, the capability for generating heaps dumps is satisfactory in all of the IBM JVM release levels. For information on generating heap dumps for IBM JDK 1.6, refer to the following document:

IBM Developer Kit and Runtime Environment, Java Technology Edition, Version 6

Related information

IBM Developer Kit and Runtim Environment - Diagnostics Guide

Generating heap dumps with Oracle HotSpot JDK

Use the following flags for heap dump generation:

- `HeapDumpOnCtrlBreak`
- `HeapDumpOnOutOfMemoryError`

For more information about analyzing heap dumps, refer to the following Oracle document:

Troubleshooting Guide for Java SE 6 with HotSpotVM

Related information

Troubleshooting Guide for Java SE 6 with HotSpot VM

Additional heap dump recommendations

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

- Heap dump generation frequently fails because the single file it generate is very large and the configuration of the supporting environment prevents regular accounts from creating such large files. Therefore, it is usually the case that you must provide some configuration to allow the account running the PolicyCenter instance 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 prompt 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 .

Heap dump generation and analysis tools

Several tools are available for heap dump generation and analysis. Both IBM and Oracle provide some tools to assist with these tasks on their respective JVMs. Other vendors also provide some tools that aim to assist with these tasks on the most common JVM platforms.

IBM-only tools for heap dump generation

IBM provides tools that can assist with heap dump analysis. You can find information on these tools on the IBM Support Assistant web site.

IBM also provides the IBM DTJF feature for Eclipse Memory Analyzer for use in analyzing IBM JVM heap dumps. You can tune the tool to use a larger heap, which is frequently necessary to analyze very large heap dumps.

Related information

IBM Support Assistant

IBM DTJF feature for Eclipse Memory Analyzer

Oracle-only tools for heap dump generation

Oracle Java Monitoring and Management Console, or JConsole, is a management tool that connects to a running Java HotSpot VM. You can trigger a heap dump by using jConsole. Refer to the following Oracle document for more information:

Using JConsole to Monitor Applications

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 following Oracle document for more information:

Java Heap Analysis Tool

Oracle jVisualVM is another multipurpose tool that you can use to analyze heap dumps. Refer to the following Oracle document for more information:

Java VisualVM

Related information

Using JConsole to Monitor Applications

Java Heap Analysis Tool

Java VisualVM

Generic tools for heap dump generation

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, the IBM DTJF feature for Eclipse Memory Analyzer, and JProbe.

JVM profiling

Java profilers are available for two main purposes:

Memory profil-ing	Identifies memory usage, and, more specifically, memory leaks due to referenced but unused objects.

CPU profiling	Helps identify programmatic hot spots and bottlenecks. This analysis might help remove the corresponding bottlenecks, thereby increasing performance.

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

- Guidewire recommends YourKit for memory profiling.
- Guidewire recommends JProfiler for CPU profiling.

To profile PolicyCenter, load the profiler agent into the PolicyCenter JVM either as it is starting PolicyCenter or by attaching the profiler agent to a running JVM. Refer to the profiler documentation for instructions.

Large object analysis

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.

Tune memory settings for application processes

If a particular process is running slowly, you can increase the amount of memory available to it by tuning a set of memory parameters. It is possible that the additional memory allocation can improve the process performance. Use the following table to identify the parameters that you can change in file `gradle.properties`, in the `PolicyCenter` installation directory.

Property	Description	Possible values	Default values
`custDistGosuCompileMaxHeapSize`	Maximum heap size for the JVM that performs Gosu compilation.	The number of gigabytes, followed by the letter g.	16g
`custDistGosuCompileMinHeapSize`	Minimum heap size for the JVM that performs Gosu compilation.	The number of gigabytes, followed by the letter g.	2g
`custDistGosudocMaxHeapSize`	Maximum heap size for the JVM that performs Gosudoc generation.	The number of gigabytes, followed by the letter g.	16g
`custDistGosudocMinHeapSize`	Minimum heap size for the JVM that performs Gosudoc generation.	The number of gigabytes, followed by the letter g.	2g
`custDistJavaCompileMaxHeapSize`	Maximum heap size for the JVM that performs Java compilation.	The number of gigabytes, followed by the letter g.	16g
`custDistJavaCompileMinHeapSize`	Minimum heap size for the JVM that performs Java compilation.	The number of gigabytes, followed by the letter g.	4g
`custDistJavaCompileSchemaSourcesMaxHeapSize`	Maximum heap size for the JVM that performs compilation of XML schema generated source for the task `genSchemaJar`.	The number of gigabytes, followed by the letter g.	16g
`custDistJavaCompileSchemaSourcesMinHeapSize`	Minimum heap size for the JVM that performs compilation of XML schema generated source for the task `genSchemaJar`.	The number of gigabytes, followed by the letter g.	4g

Property	Description	Possible values	Default values
`custDistJavaExecMaxHeapSize`	Maximum heap size for all of the tasks that run in a separate JVM, such as `genDataDictionary`.	The number of gigabytes, followed by the letter `g`.	`4g`
`custDistStudioMaxHeapSize`	The maximum amount of memory available to Guidewire Studio. See the *Configuration Guide*.	The number of gigabytes, followed by the letter `g`.	`4g`
`custDistStudioBuildProcessHeapSize`	The initial default value for the **Build process heap size** setting in the Guidewire Studio **Settings** dialog. After you run Studio for the first time, this property is ignored. See the *Configuration Guide*.	The number of megabytes.	`6000`
`org.gradle.jvmargs`	Minimum and maximum heap sizes for the JVM that runs the remaining tasks, such as codegen.	Minimum: `-Xms`, followed by the number of gigabytes, followed by the letter `g`. Maximum: `-Xmx`, followed by the number of gigabytes, followed by the letter `g`.	`-Xms1g` `-Xmx4g`

chapter 7

Geocoding

Geocoding is the process of assigning a latitude and longitude to an address. Guidewire supports geocoding in ClaimCenter, PolicyCenter, and ContactManager.

Understanding geocoding

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, PolicyCenter, and ContactManager are candidates for geocoding.

The Geocode plugin interface

To implement geocoding, PolicyCenter provides a default `GeocodePlugin` plugin interface. 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. You enable the plugin implementation in Guidewire Studio and specify the 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.

Microsoft Bing maps

Before you can use the default Bing Maps plugin implementation, your organization must have its own account, login, and application key with Microsoft Bing Maps. To obtain these items, access the Bing Maps Dev Center.

You must set up a Bing Maps account and obtain an application key. After you create an application key, the application name is arbitrary and there is no need for an application URL.

Working with Geocode batch processing

Guidewire implements geocode batch processing as a work queue. The `Geocode` writer in PolicyCenter and the `ABGeocode` writer in ContactManager use `BatchGeocode` and other criteria to filter which addresses to pass to the plugin for geocoding.

- ContactManager verifies that the `BatchGeocode` property is `true` and the `GeocodeStatus` property is `none.`
- PolicyCenter verifies the following:
  - The address is an `AccountLocation` or subtype.
  - The address is a primary address of a `UserContact`.
  - The `BatchGeocode` property is `true` and the `GeocodeStatus` property is `none`.

The geocoding process submits qualifying addresses to your implementation of the `GeocodePlugin` plugin. After the `GeocodePlugin` plugin adds geocode coordinates to an address, the geocoding process updates the address in the database.

Geocode status typelist

The `GeocodeStatus` typelist defines the set of status codes returned from the default `GeocodePlugin` plugin implementation. As the typelist is final, you cannot edit it. You access the `GeocodeStatus` typelist in Guidewire Studio in the following location:

configuration→config→Metadata→Typelist

See also

- *Integration Guide*

Related references

"Geocode Writer work queue" on page 141

Related information

Bing Maps Dev Center

Configuring geocoding

Configuring geocoding involves the following tasks:

- Enabling your implementation of the `GeocodePlugin` plugin
- Setting geocoding feature parameters
- Scheduling the `Geocode` work queue
- Configuring the number of Geocode batch processing workers

Enabling the Geocode plugin

By default, the base configuration `GeocodePlugin` plugin implementation uses the Bing Maps implementation. Guidewire disables this plugin in the base configuration. You must enable the geocode plugin before you can use the geocoding functionality. If you intend to use geocoding in multiple Guidewire applications, you must make these changes in each application separately.

Scheduling the Geocode plugin

By default, Guidewire does not schedule geocode batch processing in the base configuration. To schedule geocode batch processing, you need to uncomment the relevant section in file `scheduler-config.xml`. Guidewire recommends that you schedule the geocoding process to run during periods of minimal activity on the PolicyCenter servers.

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 geocoding batch processing workers

At the time you first start PolicyCenter, it is possible for your database to have many addresses to geocode, especially if you imported many new addresses into your production database. A large number of new addresses can cause the `GeocodePlugin` plugin implementation to take a long time to process these new addresses. Guidewire recommends that you configure the geocoding process with a sufficient number of worker instances before you start your production servers.

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

See also

- "Understanding geocoding" on page 107

About Guidewire geocoding parameters

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

Parameter	Description
`UseGeocodingInPrimaryApp`	If `true`, PolicyCenter enables searching for nearby locations in the Reinsurance Management user interface. ContactManager does not respond to this parameter. The default is `false`.
`UseMetricDistancesByDefault`	If `true`, PolicyCenter uses kilometers and metric distances instead of miles and United States distances for location searches. Set this parameter identically in Guidewire applications that use geocoding. The default is `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`. The default is 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. The default is 1000.

Enable the Geocode plugin

Procedure

1. In the PolicyCenter **Project** window, expand **configuration→config→Plugins→registry**.
2. Double-click `GeocodePlugin` to open it.
3. If the **Disabled** check box is checked, un-select the check box to enable the plugin.
4. Ensure that the **Class** field specifies the Bing Maps implementation class:
   `gw.plugin.geocode.impl.BingMapsPluginRest`
5. Under **Parameters**, specify the following:

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, refer to the following web site `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, refer to the following web site `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.

See also

- *Integration Guide*

Schedule Geocode batch processing runs

About this task

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.

Procedure

1. In Guidewire Studio, navigate to **configuration→config→scheduler** and open `scheduler-config.xml` for editing.
2. Uncomment the following section in `scheduler-config.xml` file.

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

See also

- "The work queue scheduler" on page 120
- "Understanding a work queue schedule specification" on page 121

Configure the number of worker instances for Geocode batch processing

Procedure

1. In Guidewire Studio, navigate to **configuration→config→workqueue** and open `work-queue.xml` for editing.
2. Modify the number of worker instances in the following section in file `work-queue.xml` to meet your business needs.

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

3. Update database statistics by running the Database Statistics batch process.

See also

- "Worker configuration" on page 126
- "Database Statistics work queue" on page 138

chapter 8

Administering PolicyCenter processes

PolicyCenter provides tools for configuring and managing various forms of application processes. You can schedule a process to run regularly or on demand.

See also

- “The Batch Process Info screen” on page 353
- “The Work Queue Info screen” on page 356
- *Integration Guide*

Overview of PolicyCenter processing

PolicyCenter supports two modes of processing:
- Work queue
- Batch process

Work queue

A work queue operates on a batch of items in parallel. PolicyCenter distributes work queues across all servers in a PolicyCenter cluster that have the appropriate role. In the base configuration, Guidewire assigns this functionality to the `workqueue` server role.

A work queue comprises the following components:
- A processing thread, known as a *writer*, that selects a group (batch) of business records to process. For each business record (a policy record, for example), the writer creates an associated work item.
- A queue of selected work items.
- One or more tasks, known as *workers*, that process the individual work items to completion. Each worker is a short-lived task that exists in a thread pool. Each work queue on a cluster member shares the same thread pool. By default, each work queue starts a single worker on each server with the appropriate role, unless configured otherwise.

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

Batch process

A batch process operates on a batch of items sequentially. Batch processes are suitable for low volume batch processing that achieves an acceptable throughput rate as the batch process processes items in sequence. 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.

See also

- "Server roles" on page 169
- "Work queues" on page 114
- "Batch processes" on page 117
- *Integration Guide*

Work queues

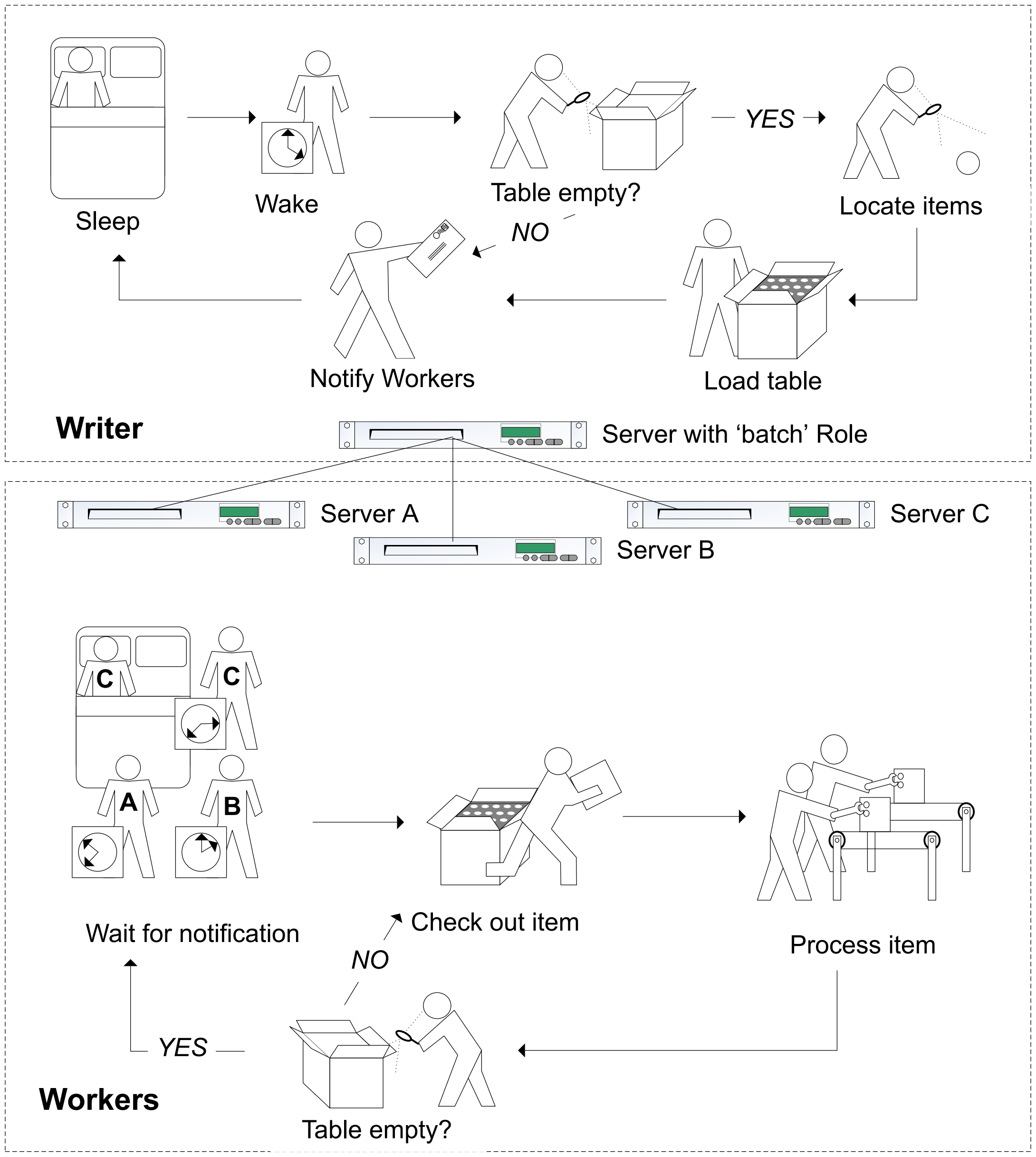
A work queue comprises the following components.

Writer	A *writer* thread selects units of work for processing and writes their identities to a work queue.
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 tasks* that check out work items from the work queue and process them to completion. By default, each work queue starts a single worker on each cluster member with the `workqueue` role, unless configured otherwise.

Starting the writer initiates a run of processing on a work queue. The work is complete if 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.



Work queue writers

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. The specific configuration of each work queue determines how the work queue creates and processes work items.

The following workflow examples provides a simple description of how PolicyCenter work queues operate.

Work items remaining

If work items remain from a previous batch, the following sequence of events occur:

1. The writer thread notifies the workers that they have work to process.
2. PolicyCenter terminates the writer thread.

No work items remaining

If no work items remain from a previous batch, the following sequence of events occurs:

1. The writer thread selects items for a new batch.
2. The writer writes the identifiers of the selected items to the work queue table.
3. The writer notifies the workers that they have work to process.
4. PolicyCenter terminates the writer thread.

Work queue workers

Typically, each work queues shares the standard work item table (`StandardWorkQueue`) for its work items. However, a worker task operates only on work items that its associated writer inserts into the table. For example, suppose that PolicyCenter distributes the Activity Escalation work queue across a Guidewire cluster, with six workers operating on three different servers. Those workers work only on activity escalation items in the standard work item table. Typically, you configure work queues for multiple workers. Thus, typically, some number of workers operate throughout the day on work items in the standard work queue table.

At specified intervals (defined through the `maxpollinterval` attribute on `worker` in `work-queue.xml`), a worker awakens and checks the work item table for work items from its associated writer. If a worker 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`. The value of the `Status` attribute can be any one of the following: • `available` • `checkedout` • `failed`
`LastUpdateTime`	Set to the time at which the worker checks out the work item.
`CheckedOutBy`	Set to the worker.

After it checks out a quota of work items, the worker task 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.

Note: In rare cases, it is possible for PolicyCenter to notify a worker of work, but, the worker finds no work is available after it awakens. For example, for small batch runs, a worker can check out all items in the batch with its first check out quota of items. This action can occur between the time the writer notifies the workers and other workers awaken. If a worker awakens and finds no work items, the worker goes back to sleep.

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. Access schedule configuration file `schedule-config.xml` in Guidewire Studio at the following location:

configuration→config →scheduler

Worker tasks awaken much more frequently than their writers start. One writer awakens at least as frequently as a specified maximum polling interval, if not more frequently.

You do not schedule worker tasks. Instead, they awaken in response to notification from the writer or upon expiration of the polling interval. After a worker awakens, if there are work items to process, it processes up to the number of `batchsize` items, as defined in `work-queue.xml`. 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 maximum allowed number of workers are active.

You configure the number of workers, polling interval, and batch size in `work-queue.xml`. Access work queue configuration file `work-queue.xml` in Guidewire Studio at the following location:

configuration→config→workqueue

You can view and manage work queues from the Server Tools `Work Queue Info` screen in PolicyCenter, if you have the appropriate administrative privilege.

See also

- “The work queue scheduler” on page 120
- “Performing custom actions after a process completes” on page 128
- “The Work Queue Info screen” on page 356

Batch processes

PolicyCenter distributes batch processes across all PolicyCenter cluster members that have the `batch` server role. Each server with the `batch` role also has a batch process lease manager that acquires and manages the batch process leases on that server. In this context, a lease represents a single run of a single batch process.

Available servers with the `batch` role compete for available batch processing leases. After a server acquires a lease, that server runs the batch process to completion.

How aggressively the cluster servers compete with each other depends on how many batch processes each one is individually already running. Those servers running fewer or no batch processes are more likely to acquire a new batch process lease than other servers already busy running processes. It is possible to configure this behavior.

For scheduled batch processes, a scheduler component, running on a cluster member with the `scheduler` role, decides to start a batch process according to the published schedule. The scheduler first creates a new lease for the batch process in the database. All cluster members with the `batch` server role then compete to acquire that lease. The cluster member that wins the competition starts the batch process.

The Batch Process Info screen

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 Server Tools **Batch Process Info** screen in PolicyCenter, if you have the appropriate administrative privilege. From this screen, you can view the batch process schedule, if any, and start or stop a scheduled batch process.

Batch process caching

Batch processes that run concurrently on the same server 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. Concurrent data change exceptions can occur also if multiple batch processes (on the same or different servers) attempt to update the same cached entity instances. This requires one or the other batch process to retry an item, leading to further performance degradation.

See also

- “Working with work queue writers and batch processes” on page 118
- “The work queue scheduler” on page 120
- “Component lease management” on page 187
- “Batch process prioritization” on page 194
- “The Batch Process Info screen” on page 353

Working with the different processing types in Studio

To access information about the PolicyCenter processing types, open the following files in Studio:

- `BatchProcessType.ttx` – Provides the name and codes for each processing type.
- `work-queue.xml` – Provides the name of the backing class and number of work instances for a processing type.
- `scheduler-config.xml` – Provides the schedule for any process that is Schedulable.

The number of worker tasks is important if zero (0). Setting the number of worker tasks to 0 disables the work queue as there are no workers to perform the work. To enable the work queue, set the number of workers to greater than zero.

Working with work queue writers and batch processes

You can run many batch processes, including writers for work queues, from PolicyCenter or from the command prompt.

How to run a writer from the Work Queue Info screen

It is possible to run a writer for a work queue from the Server Tools **Work Queue Info** screen. PolicyCenter enables the **Run Writer** button on this screen for all work queue types that belong to the `BatchProcessTypeUsage` category `UIRunnable`.

To access the **Work Queue Info** screen, you must have the `internaltools` permission. 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 the **Work Queue Info** screen.

How to run a batch process from the Batch Process Info screen

It is possible to run batch processes from the Server Tools **Batch Process Info** screen in PolicyCenter. PolicyCenter enables the **Run** button on this screen for all batch process types that belong to the `BatchProcessTypeUsage` category `UIRunnable`.

To access the **Batch Process Info** screen, you must have the `internaltools` permission. 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** screen.

How to terminate a writer or batch process from the command prompt

It is possible to terminate some in-progress processes, including writers for work queues, from a command prompt. To determine if it is possible to terminate an in-progress batch process, consult the reference topic for that particular batch processing type. The information for each individual batch process includes whether it is a single phase or multiphase process. You can only stop those processes listed as being multiphase.

Note: It is not possible to terminate a *single phase* batch process. Single phase processes run in a single transaction. Thus, there is no convenient place to terminate the process.

See also

- “Work queues and batch processes, a reference” on page 130
- “The Batch Process Info screen” on page 353
- “The Work Queue Info screen” on page 356
- “maintenance_tools command” on page 426
- *Integration Guide*

Run a writer from PolicyCenter

Start or stop a work queue from the **Work Queue Info** screen.

Before you begin

You must have the `internaltools` user permission to access the **Work Queue Info** screen.

About this task

It is possible to run writers for work queues from either the **Work Queue info** screen or the **Batch Process Info** screen.

Procedure

1. Log in to PolicyCenter.
2. Press `ALT+SHIFT+T` to display the **Server Tools** tab.
3. Navigate to **Work Queue Info**.

4. Click **Run Writer** in the **Actions** column of the work queue that you want to run.

Run a batch process from PolicyCenter

Start, or stop, a batch process from the **Batch Process Info** screen.

Before you begin

You must have the `internaltools` user permission to access the **Batch Process Info** screen.

Procedure

1. Log in to PolicyCenter.
2. Press `ALT+SHIFT+T` to display the **Server Tools** tab.
3. Navigate to **Batch Process Info**.
4. Click **Run** in the **Action** column of the batch process that you want to run.

Run a writer or batch process from the command prompt

Start a process using a `maintenance_tools` command option.

About this task

It is possible to run many processes, including writers for work queues, from a command prompt.

Procedure

1. Start the PolicyCenter server if it is not already running.
2. Open a command prompt.
3. Navigate to the following location in the PolicyCenter installation directory:

```
admin/bin
```

4. Run the following command:

```
maintenance_tools -password password -startprocess process
```

   The PolicyCenter administration command prompt tools all require that you enter the password of an administrative user for the tool to work. The use of a user name is optional.

   For the *`process`* value, specify a valid process code. For the process code for each process, consult the reference topic for the individual batch processing type.

See also

- "Work queues and batch processes, a reference" on page 130

Terminate a writer or batch process from the command prompt

Stop a process using a `maintenance_tools` command option.

About this task

It is not possible to use following procedure to terminate the operation of the `table_import` command.

Procedure

1. Start the PolicyCenter server if it is not already running.
2. Open a command prompt.

3. Navigate to the following location in the PolicyCenter installation directory:

```
admin/bin
```

4. Run the following command:

```
maintenance_tools -password password -terminateprocess process
```

   The PolicyCenter administration command prompt tools all require that you enter the password of an administrative user for the tool to work. The use of a user name is optional.

   For the *process* value, specify a valid process code. For the process code for each process, consult the reference topic for the individual batch processing type.

See also

- "Work queues and batch processes, a reference" on page 130

Check status of a writer or batch process from the command prompt

Check the status of a process using a `maintenance_tools` command option.

About this task

It is possible to check the status of processes, including writer processes for work queues, from a command prompt.

Procedure

1. Start the PolicyCenter server if it is not already running.
2. Open a command prompt.
3. Navigate to the following location in the PolicyCenter installation directory:

```
admin/bin
```

4. Run the following command:

```
maintenance_tools -password password -processstatus process
```

   The PolicyCenter administration command prompt tools all require that you enter the password of an administrative user for the tool to work. The use of a user name is optional.

   For the *process* value, specify a valid process code. For the process code for each process, consult the reference topic for the individual batch processing type.

Result

For work queues, executing this command returns the status of the writer process. The command does not check whether any work items remain in the work queue. Thus, it is possible for the process status to report as being complete after the writer finishes adding items to the work queue. However, it is possible that there are work items that need processing that remain in the work queue.

See also

- "Work queues and batch processes, a reference" on page 130

The work queue scheduler

The PolicyCenter scheduler launches many processes, including writer processes for work queues, according to a schedule defined in `scheduler-config.xml`. Access this file in Guidewire Studio at the following location:

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` attribute sets the process to run. The `env` attribute is an optional attribute that specifies the environment in which the schedule definition for the process applies. The *`schedule_attributes`* value is a valid schedule specification.

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 process runs in hours as opposed to minutes. This is because some batch processes require a lot of server resources. Schedule such processes to wake infrequently or at times that the server is less busy, such as late at night or very early in the morning.

You may want to schedule some PolicyCenter processes to run periodically throughout the business day. For example, the default configuration of PolicyCenter schedules the `ActivityEsc` work queue to run every 30 minutes. Exclude running such processes periodically during your nightly processing window. Instead, wait until the end of the processing window to run them. For example, schedule the `ActivityEsc` process to run every 30 minutes except during your nightly processing window. Alternatively, run such processes at prescribed places in your chain of nightly processes.

The PolicyCenter scheduler uses the PolicyCenter server time for reference.

See also

- "Understanding a work queue schedule specification" on page 121

Understanding a work queue schedule specification

The `<CronSchedule>` element in file `scheduler-config.xml` describes when PolicyCenter is to run the process.

```
<CronSchedule schedule_attributes/>
```

Use this element to define a *`schedule_attributes`* value to specify the exact timing, such as once every hour or every night at a certain time. The *`schedule_attributes`* value 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"`

IMPORTANT If you do not provide a value for a defined schedule attribute, the scheduler uses its default value in determining the work queue schedule. For example, if you do not specify a value for the `hours` attribute, the scheduler assumes a value of `*` and PolicyCenter runs the work queue process every hour. Thus, Guidewire recommends that you provide a value for each scheduler attribute. If you do not provide a value for a specific attribute, carefully review that attribute's default value and determine if the default value meets your business needs.

Useful attribute characters

Along with the standard values listed, there are some special characters that give you more flexible options.

Character	Meaning
`*`	Indicates all values. For example, `minutes="*"` means run the process every minute.
`?`	Indicates 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`	Specifies the last day. Used only for `dayofmonth` and `dayofweek` attributes. See the examples for clarification.
`W`	Specifies the nearest weekday, use only with `dayofmonth`. For example, if you specify `1W` for `dayofmonth`, and that day is a Saturday, the trigger then fires 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"`.
`#`	Specifies 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 Wednesday in the month).

These represent only some of the values that you can use for setting schedule.

Scheduler examples

The following table lists a few examples of how to work with the `<CronSchedule>` element.

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.

Example	Description
`<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.

The Quartz Enterprise Job Scheduler

These characters represent only some of the values that you can use for setting a schedule. Guidewire bases the PolicyCenter scheduler on the open source Quartz Enterprise Job Scheduler. The scheduler 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`.

Related information

Quartz Documentation

Determine if it is possible to schedule a batch process

Use the **Batch Process Info** screen to determine if it is possible to schedule a batch process.

About this task

It is possible to schedule many batch processes, including work queue writers. It is not possible, however, to schedule all batch processes.

Procedure

1. Log into PolicyCenter as an administrative user.
2. Press `ALT+SHIFT+T` to access Server Tools.
3. Navigate to the **Batch Process Info** screen.
4. Select **Schedulable** from the processes drop-down filter.

   PolicyCenter displays only those batch processes, including work queue writers, that it is possible to schedule in file `scheduler-config.xml`.

View a batch process schedule in PolicyCenter

Review information in the **Batch Process Info** screen.

Procedure

1. Log in to PolicyCenter as an administrative user.
2. Press `ALT` + `SHIFT` + `T` to access Server Tools.
3. Navigate to the **Batch Process Info** screen.
4. Click the **Next Scheduled Run** column header to sort processes by schedule.

   If there is no current schedule for a process, the **Next Scheduled Run** field is blank.

Scheduling batch processes for specific environments

You can define batch process schedules for different environments. To specify an environment for a process schedule, include the `env` attribute on the `<ProcessSchedule>` element in the file, `scheduler-config.xml`.

```
<ProcessSchedule process="process_code" env="environment1, environment2, …">
  <CronSchedule schedule_attributes/>
</ProcessSchedule>
```

In this way, you can have different schedules for a process based on environment.

Disabling the PolicyCenter scheduler

It is possible to disable the internal scheduler by setting the `SchedulerEnabled` configuration parameter to `false` in file `config.xml`. If you do so, then you need to implement your own mechanism for running PolicyCenter processes. For example, you can use your own scheduling application to trigger process execution along with one of the following:

- The `startBatchProcess` method on the `MaintenanceToolsAPI` web service
- The `maintenance_tools -startprocess` *`process`* command option

Configuring work queues

In working with work queues, there are multiple general areas that you can configure.

Configuration area	Configuration file	More information
Work queue scheduling	`scheduler-config.xml`	"The work queue scheduler" on page 120
Workers and work queues	`work-queue.xml`	"The work queue configuration file" on page 124
Work queue configuration parameters	`config-xml`	*Configuration Guide*

The work queue configuration file

You may want to modify the configuration of Guidewire-provided work queues to improve performance. You configure attributes of a work queue and its workers in file `work-queue.xml`. For custom work queues, you must modify `work-queue.xml` to enable your work queue to operate.

File `work-queue.xml` contains one top-level element, which is `<work-queues>`. This element has one required attribute, `defaultServer`. In the base configuration, Guidewire sets the value of this attribute to `workqueue`.

```
<work-queues xmlns="http://guidewire.com/work-queue" defaultServer="#workqueue">
```

Note: The hash mark in front of `workqueue` (`#workqueue`) indicates that the value that follows the hash mark is a server role and not a server ID.

Attribute `defaultServer` requires a value. There is no default. The PolicyCenter server refuses to start if you do not provide a value for this attribute. The server also refuses to start if you set `defaultServer` to a role that does not exist in `<registry>` element in `config.xml`.

Work queue definitions

Within the top-level `<work-queues>` element in `work-queue.xml`, use subelement `<work-queue>` to define individual work queues.

```
<work-queue workQueueClass="string" progressinterval="decimal">
  <worker instances="integer" throttleinterval="decimal" env="string1, string2, …" server="string"/>
</work-queue>
```

The `<work-queue>` subelement has attributes to configure a named work queue in general. The `<worker>` subelement has attributes to configure worker tasks on specific servers. You can declare as many workers as you want for a work queue by specifying on which servers the workers run. You can declare a set of possible environments for a worker by specifying a value or multiple comma-separated values for the `env` attribute.

Access `work-queue.xml` in Guidewire Studio at the following location:

configuration→config →workqueue

See also

- "General work queue configuration" on page 125
- "Worker configuration" on page 126

General work queue configuration

The `<work-queue>` element in `work-queue.xml` contains attributes for configuring the general characteristics of a work queue.

Attribute	Description
Required attributes	
`progressinterval`	The `progressinterval` value is the amount of time, in milliseconds, that PolicyCenter allots for a worker to process the number of `batchsize` work items. If the time a worker has held a batch of items exceeds the value of `progressinterval`, then PolicyCenter considers the work items to be orphans. PolicyCenter reassigns orphaned work items to a new worker instance. The `progressinterval` value must be greater than the time to process the slowest work item, or that work item never completes. Guidewire recommends that you set the `progressinterval` value greater than the processing time for an entire `batchsize` of work items: • If a worker takes more time than the time specified by `progressinterval` to processes its assigned work items, PolicyCenter reverts the remaining work items to `available` from `checkedout`. • If many worker batches take longer than the time specified by `progressinterval`, the repeated checking out and reverting to available of work items can have a negative impact on performance.
`workQueueClass`	(Required) The `workQueueClass` value must be one of the following: • A Guidewire-provided work queue class listed in the base configuration version of `work-queue.xml` • A custom work queue class derived from Gosu class `WorkQueueBase` You cannot configure Guidewire-provided batch processes or custom batch processes derived from the Gosu class `BatchProcessBase`.
Optional attributes	
`blockWorkersWhenWriterActive`	If the work queue workers start execution before the work queue writer completes writing work items to the work queue, it can possibly cause performance issues under certain circumstances. If set to `true`, PolicyCenter blocks the work queue workers from acquiring new work items until the writer completes writing work items to the queue. After the writer completes writing any new work items, the workers automatically start acquiring work items again. The default is `false`. Only enable this attribute for the work queues for which you require this capability. Guidewire recommends that you consider setting this attribute to `true` if the work queue writer can run for extensive periods of time due to the work load generated.
`logRetryableCDCEsAtDebugLevel`	If the value of `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 to indicate that the error is non-fatal. PolicyCenter logs any CDCE that pushes the retry count over the value of `retryLimit`, or the value of `workItemRetryLimit` if `retryLimit` is not set, at the ERROR level.
`retryInterval`	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`	The number of times PolicyCenter retries a work item that threw an exception or that 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. IMPORTANT: Guidewire generally recommends that you increase, never decrease, the number of retries for a work queue.

Worker configuration

The use of the `<worker>` element in `work-queue.xml` is optional. However, in actual practice, it is necessary for there to be at least one `<worker>` element for each `<work-queue>` element for the work queue to operate properly. The `<worker>` element contains an `instances` attribute that has a default value of 1. Without a `<worker>` element to provide this default, the processing logic does not allocate any workers for the work queue.

All of the following attributes are optional.

Attribute	Description
`instances`	The number of workers to create. By default, PolicyCenter sets the values of this attribute to 1. If a worker wakes up and detects work items, it checks out those work items from the work queue. If there are more work items than the value specified by the `batchsize` attribute, the worker starts another worker. Each new worker checks out up to the maximum `batchsize` number of work items. If there are more work items remaining, the new worker starts another worker. The creation of workers continues until the number of workers reaches the maximum limit of workers as specified by the `instances` attribute.
`maxpollinterval`	How often a worker wakes up automatically and queries for work items, even if the worker receives no notification. You might need to increase the value of `maxpollinterval` to prevent excessive numbers of queries for work items. The default value of `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 tasks process work items as rapidly as possible. To reduce the CPU load, set the value of `throttleinterval` to a positive non-zero 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 value for `batchsize` is 10.
`env`	The one or more environments in which this particular worker configuration is active. To specify multiple values for the `env` attribute, use a comma-separated list.
`server`	The `serverid` of the server on which this particular worker is active.

See also

- For information about the definition of the `env` and the `serverid` values in the cluster registry in `config.xml`, see “Understanding the configuration <registry> element” on page 60.

Worker task management

An *executor* manages the worker tasks on each cluster server with the appropriate role. In the base configuration, Guidewire assigns this functionality to the `workqueue` server role. Each server with the `worqueue` role creates one executor for each work queue on that server.

Each work queue executor periodically creates a worker. (The executor can also create a worker upon receiving a notification from the writer.) This worker checks the work queue for items to process. If necessary, the initial worker creates an additional worker if there is more work to process than it can handle. This new worker can also create a worker if there is still more work to process. After there is no more work to process, all active workers stop.

It is possible to control the maximum number of workers, for all work queues on a server, by setting the value of configuration parameter `WorkQueueThreadPoolMaxSize` in `config.xml`. It is possible to set this value individually on each PolicyCenter server in the cluster.

See also

- "General work queue configuration" on page 125
- "Worker configuration" on page 126
- "Work queues and server roles" on page 127
- *Configuration Guide*

Work queues and server roles

In the base configuration, Guidewire assigns work queue functionality to servers with the `workqueue` role. File `work-queue.xml` associates the `workequeue` server role with the application work queue functionality.

```
<work-queues xmlns="http://guidewire.com/work-queue" defaultServer="#workqueue" />
```

Note: The hash mark in front of `workqueue` (`#workqueue`) indicates that the value that follows the hash mark is a server role and not a server ID.

The `workqueue` role is merely the default role, however. You are free to create and assign new work queue management roles. You can also use server roles to enable or disable certain work queues on a specific PolicyCenter server.

Defining a new work queue role

You define server roles using the `roles` attribute on the `<registry>` element in file `config.xml`. In the base configuration, Guidewire defines the following server roles:

```
<registry roles="batch, workqueue, scheduler, messaging, startable, ui" />
```

To add a specialized work queue role, say, one to use in managing activity work queues, you need merely to add the new server role to the list of roles:

```
<registry roles="batch, workqueue, activityworkqueue, scheduler, messaging, startable, ui" />
```

See also

- "Defining a new server role" on page 64

Assigning a work queue to specific PolicyCenter cluster servers

Suppose that you want to assign the management of the activity work queues in the PolicyCenter cluster to a subset of the cluster servers with the `activityworkqueue` role. By default, PolicyCenter distributes all other work queues to those servers with the default `workqueue` role.

For example, in file `work-queue.xml`, you define the following:

```
<?xml version="1.0"?>
<work-queues xmlns="http://guidewire.com/work-queue" defaultServer="#workqueue">
  <work-queue workQueueClass="com.guidewire.pl.domain.escalation.ActivityEscalationWorkQueue" … >
    <worker server="#activityworkqueue"/>
  </work-queue>
  <work-queue workQueueClass="com.guidewire.pl.domain.geodata.geocode.GeocodeWorkQueue" … >
    <worker/>
  </work-queue>
</work-queues>
```

In this example:

1. If a server has the `workqueue` role only, then that server:
   a. Starts an executor for the `GeocodeWorkQueue` work queue.
   b. Does not start an executor for the `ActivityEscalationWorkQueue` work queue.
2. If a server has the `activityworkqueue` role only, then that server:
   a. Starts an executor for `ActivityEscalationWorkQueue` work queue.
   b. Does not start an executor for the `GeocodeWorkQueue` work queue.
3. If a server has both the `activityworkqueue` and `workqueue` roles, then that server starts executors for both work queues.
4. If a server has neither the `activityworkqueue` nor the `workqueue` role, then the server does not start an executor for either of these work queues.

Performing custom actions after a process completes

You can use Process Completion Monitor processing 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.

Process Completion Monitor processing runs at schedulable intervals and examines the `ProcessHistory` table for all completed work queues and batch processes.

For each completed work queue that it finds, Process Completion Monitor:

- Determines if all the work items in that work queue have either completed or failed.
- Calls the `IBatchCompletedNotification` plugin implementation on a process if the process is complete and has no remaining available or checked-out work items.
- Sets `ProcessHistory.NOTIFICATIONSENT` to `true` to invoke the `IBatchCompletedNotification` plugin implementation a single time only for any given process.

The `IBatchCompletedNotification` interface has a `completed` method that you can override to perform specific actions if a work queue or batch process finishes a batch of work. The parameters of the `completed` method are the `ProcessHistory` entity and the number of failed items. PolicyCenter considers work queue processing as complete if no work items remain on the queue, other than work items that failed. PolicyCenter considers a batch process as complete if the process stopped and its process history is available.

See also

- “Schedule the Process Completion Monitor batch process” on page 128
- “Implement the IBatchCompletedNotification interface” on page 129
- “Register a custom batch notification plugin” on page 129
- “Process Completion Monitor batch process” on page 147

Schedule the Process Completion Monitor batch process

The Process Completion Monitor process runs at schedulable intervals and examines the `ProcessHistory` table for all completed work queues and batch processes.

About this task

In the base configuration, Guidewire does not schedule the Process Completion Monitor process. To enable this process, you need to add the process to file `scheduler-config.xml`.

Procedure

1. In the PolicyCenterStudio **Project** window, expand **configuration→config→scheduler**.
   a. Open `scheduler-config.xml`.
   b. Add the following `<ProcessSchedule>` element:

```
<ProcessSchedule process="ProcessCompletionMonitor">
  <CronSchedule minutes="*/5"/>
</ProcessSchedule>
```

2. Save your changes.

See also

- "Understanding a work queue schedule specification" on page 121

Implement the IBatchCompletedNotification interface

The Process Completion Monitor calls the `IBatchCompletedNotification` plugin implementation on a process if the process is complete and has no remaining available or checked-out work items.

Procedure

1. In the PolicyCenterStudio **Project** window, expand **configuration→gsrc**.
2. Do one of the following:
   - If a package for your plugin implementation classes already exists within **gsrc**, navigate to that package, then skip to "Implement the IBatchCompletedNotification interface" on page 129.
   - If a package for your plugin implementation classes does not exist, continue to "Implement the IBatchCompletedNotification interface" on page 129.
3. Right-click **gsrc**, then click **New→Package**.
4. Enter a package name, such as `workqueue`.
5. Right-click your implementation class package and click **New→Gosu Class**.
6. Enter the name `IBatchCompletedNotification` for the gosu class.
7. Click **OK**.
8. Define your Gosu class, using the following framework:

```
package myCompany.plugin.workqueue
uses gw.plugin.workqueue.IBatchCompletedNotification

 class IBatchCompletedNotification implements IBatchCompletedNotification {

   construct() { }

   override function completed(batch : ProcessHistory, numFailed : int) {

     //do something

    return
  }
}
```

9. Save your work.

Register a custom batch notification plugin

After you create your `IBatchCompletedNotification` plugin implementation, you need to register the plugin with PolicyCenter.

Procedure

1. In the PolicyCenterStudio **Project** window, expand **configuration→config→Plugins**.
2. Right-click **registry** and click **New→Plugin**.
3. In the **Plugin** dialog, enter `IBatchCompletedNotification` for the name of your plugin.
4. In the **Plugin** dialog, click ...
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 that you entered.
7. Click the **Add Plugin** + icon and select **Add Gosu Plugin**.
8. For **Gosu Class**, enter your class, including the package.
9. Save your changes.

See also

- *Configuration Guide*

Troubleshooting work queues

It is possible for a work queue worker to encounter a problem that causes it to fail, before the worker completes all the items it checked out from the queue. For example, it is possible for a server to die, killing its workers in the middle of processing. This action can result in orphan work items. A work item becomes an orphan if a worker has an item checked out but does not complete processing the item within an allotted amount of time. The `progressinterval` attribute on the `<worker>` element in `work-queue.xml` defines this time span.

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 can occur while a worker waits for an external server to return a value. If the log contains these type of timeouts, increase the `progressinterval` value for the work queue in `work-queue.xml` to give workers more processing time.

Sometimes, a problem inherent in the item itself causes the processing of the item to fail. For example, a batch process throws an exception. In such cases, the worker stops processing the item and goes on to the next. The item becomes an orphan 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 configured for the work queue. If a work item exceeds the limit of processing attempts, PolicyCenter changes the status of the work item to `failed`. Workers ignore items with a status of `failed` and no longer attempt to process them.

See also

- “Purge Failed Work Items batch process” on page 149

Work queues and batch processes, a reference

In the base configuration, PolicyCenter provides a number of work queues and batch processes.

See also

- “Working with work queue writers and batch processes” on page 118
- “Terminate a writer or batch process from the command prompt” on page 119
- “The work queue scheduler” on page 120
- “Unused and internal batch processes” on page 158
- “The Batch Process Info screen” on page 353
- “The Work Queue Info screen” on page 356

Account Holder Count work queue

Code	`AccountHolderCount`
Categories	`UIRunnable`, `Schedulable`

Implementation	Work queue
Schedule	Not scheduled
Class	`AccountHolderCountWorkQueue.java`

On `Contact` objects, the `AccountHolderCount` values sometimes are incorrect. The Account Holder Count work queue finds `Contact` objects with incorrect `AccountHolderCount` values, and updates the value. If the value of `AccountHolderCount` is set incorrectly, it can affect search performance.

Based on the number of contacts that need updating, decide whether to run Account Holder Count manually or automatically as a scheduled batch process.

- If the number of contacts to update is relatively small and does not change often, consider running it manually. Periodically check the data distribution in case something later causes `Contact.AccountHolderCount` fields to be incorrect.
- If the number of contacts to update is large, or if contacts regularly have incorrect `AccountHolderCount` values, consider scheduling Account Holder Contact to run on a regular basis.

Determine the number of incorrect AccountHolderCount contacts

About this task

In the base configuration, Guidewire does not schedule the Account Holder Count process. To decide whether to schedule the process or run it manually, do the following:

Procedure

1. Log into Policy using an administrative account.
2. Press `ALT+SHIFT+T` to open the Server Tools screen.
3. Navigate to **Info Screens→Data Distribution**.
4. In the **Collect distributions for all tables** field, select **Specify tables**.
5. Add the `pc_contact` table.
6. Click **Submit Data Distribution Batch Job**.

   The `pc_contact` data distribution table reports the *Number of Contacts with incorrect AccountHolderCount field*.

See also

- “The Data Distribution screen” on page 374

Account Withdraw Evaluation work queue

Code	`AccountWithdraw`
Class	`AccountWithdrawWorkQueue.gs`
Categories	`UIRunnable`, `Schedulable`
Implementation	Work queue
Schedule	Once a day, at 2 a.m.

The Account Withdraw Evaluation work queue marks the account status as withdrawn (`Withdrawn`) if:

- There are no policies associated with the account.
- The `Account.CreateTime` or `Account.OriginationDate` is older than a configurable number of months in the past. The `AccountsWithdrawnAfterMonths` parameter in `config.xml` specifies the number of months. In the base configuration, this parameter is set to 37 months.
- There are no open activities associated with the account.

Activity Escalation work queue

Code	`ActivityEsc`
Categories	`UIRunnable`, `Schedulable`
Implementation	Work queue
Schedule	Every 30 minutes
Class	`ActivityEscalationWorkQueue.java`

The Activity Escalation work queue finds activities that meet certain escalation criteria and marks the activity for escalation. The work queue logic looks for activities that meet each of the following criteria:

- The activity has an escalation date.
- The escalation date is prior to today's date.
- PolicyCenter has not previously escalated the activity.

If the Activity Escalation work queue finds an activity that meets all the criteria, it marks the activity as escalated and calls the activity escalation rules to determine any actions.

If you set your escalation deadline in days, then there is no reason to run activity escalation more than daily. However, if your escalation deadline is shorter, then run this process more frequently to take action on overdue activities in a timely manner. By default, PolicyCenter runs Activity Escalation work queue every 30 minutes. As indicated, you can change this schedule as needed.

See also

- *Application Guide*
- *Configuration Guide*
- *Rules Guide*

Apply Pending Account Data Updates work queue

Code	`ApplyPendingAccountDataUpdates`
Categories	`UIRunnable`, `Schedulable`
Implementation	Work queue
Workers	10
Schedule	Once a day, at 12:10 a.m.
Class	`ApplyPendingAccountDataUpdatesWorkQueue.gs`

The Apply Pending Account Data Updates work queue applies any pending updates to account data.

See also

- *Application Guide*

Archive Policy Term work queue

Code	`ArchivePolicyTerm`
Categories	`UIRunnable`, `Schedulable`
Implementation	Work queue
Schedule	Not scheduled
Class	`ArchivePolicyTermWorkQueue.java`

The Archive Policy Term work queue archives policy terms. The work queue calls the `IPCArchivingPlugin` implementation to determine whether a policy is eligible for archiving.

For a policy period to be eligible for archiving, the server time must have reached the `PolicyTerm.NextArchiveCheckDate` date.

The Archive Policy Term work queue makes large changes to database tables. After running the Archive Policy Term work queue, Guidewire recommends that you update database statistics. Updating database statistics enables the optimizer to pick better queries based on more current data.

See also

- "Understanding database statistics" on page 285
- "The Archive Info screen" on page 362
- *Application Guide*
- *Configuration Guide*
- *Integration Guide*

Archive Reference Tracking Synchronization work queue

Code	`ArchiveReferenceTrackingSync`
Categories	`Schedulable`, `UIRunnable`
Implementation	Work queue
Schedule	Not scheduled
Class	`ArchiveReferenceTrackingSyncWorkQueue.java`

You run the Archive Reference Tracking Synchronization work queue once to find all references from any archived documents to any object instances in the entity graph. This work queue creates a table of archived objects to make it faster to make this determination.

See also

- For a full description of the `ArchiveReferenceTrackingSync` work queue, see the *Configuration Guide*.

Asynchronous Quoting work queue

Code	`AsyncQuoting`
Categories	None
Implementation	Work queue
Class	`AsyncQuotingWorkQueue.gs`

The Asynchronous Quoting work queue quotes the current policy period asynchronously. This work queue processes `AsyncQuotingWorkItem` work items. `AsyncQuotingWorkItem` is an entity that delegates to `WorkItem`. This work item has properties for the following:

`RequestingUser`
The user that requested the quote

`PolicyPeriod`
The policy period to quote

Audit Task work queue

Code	`AuditTask`
Categories	`UIRunnable`, `Schedulable`
Implementation	Work queue
Schedule	Once a day, at 5:00 a.m.
Class	`AuditTaskMonitorWorkQueue.java`

The Audit Task work queue starts scheduled audits.

See also

- *Application Guide*

Bound Policy Exception work queue

Code	`BoundPolicyException`
Categories	`UIRunnable`, `Schedulable`
Implementation	Work queue
Schedule	Not scheduled
Class	`BoundPolicyExceptionWorkQueue.java`

The Bound Policy Exception work queue runs policy exception rules on every bound `PolicyPeriod` that has not had exception rules run on it for more than a defined number of days.

Parameter `BoundPolicyThresholdDays` in `config.xml` defines the minimum number of days that must pass before PolicyCenter runs the Exception rules again on a bound `PolicyPeriod`. To be eligible for processing, the `PolicyPeriod` must be either in force or expired within the last year.

See also

- *Rules Guide*
- *Configuration Guide*

Bulk Submission Job batch process

Code	`BulkSubmission`
Class	`BulkSubmission.gs`
Categories	`UIRunnable`, `Schedulable`
Implementation	Batch process

Schedule	Not scheduled

Bulk Submission Job batch processing is part of Smart Communications for PolicyCenter. Bulk Submission Job batch processing enables creation of documents asynchronously using SmartCOMM templates using the SmartCOMM Bulk API. In the base configuration, Bulk Submission Job passes the location of the files to SmartCOMM. SmartCOMM processes the files and creates the documents.

Clean Up Account Contact Role Table work queue

Code	`CleanupAccountContactRole`
Class	`CleanupAccountContactRoleWorkQueue.gs`
Categories	`UIRunnable`, `Schedulable`
Implementation	Work queue
Schedule	Saturday, at 4 a.m.

The Clean Up Account Contact Role Table work queue deletes retired `AccountContactRole` entity instances and any associated `AccountContactRoleReplacement` entity instances. The work queue marks `AccountContactRole` entity instances that are no longer in use as `Retired`.

Clean Up PurgedRootInfo work queue

Code	`CleanupPurgedRootInfo`
Categories	`Schedulable`
Implementation	Work queue
Schedule	Not scheduled
Class	`PurgeOldPurgedRootInfoWorkQueue.gs`

When purging a `Contact`, `Account`, `Policy`, `PolicyTerm`, or `PolicyPeriod`, PolicyCenter adds a `PersonalDataPurge` event and adds a corresponding `PurgedRootInfo` entity. Guidewire DataHub can then extract the data from `PurgedRootInfo`.

The `CleanupPurgedRootInfo` work queue removes `PurgedRootInfo` entities whose `PurgeType` is `PersonalData`. To be considered, the entities must have a purge date older than the number of days specified in the configuration parameter `KeepPurgedRootForDays`. The default value of this parameter is 180 days.

You can also open Guidewire Studio and schedule the work queue in `scheduler-config.xml`. In the base configuration, Guidewire comments out this entry. You can remove the comments from the following entry and provide your own settings as well.

```
<!-- Delete old PurgedRootInfo entities that have a purged date older than KeepPurgedRootsForDays -->
<!--  <ProcessSchedule process="CleanupPurgedRootInfo">
    <CronSchedule dayofmonth = "?" dayofweek="MON" hours="4"/>
  </ProcessSchedule>
-->
```

Clear Policy Renewal Check Dates batch process

Code	`PolicyRenewalClearCheckDate`
Categories	`APIRunnable`, `MaintenanceOnly`
Implementation	Batch process

Stoppable	No (single phase process)
Schedule	Not schedulable
Class	`PolicyRenewalClearCheckDate.gs`

The Clear Policy Renewal Check Dates 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. In other words, the server must be a run level of `NO_DAEMONS` or lower.

Only run this process if a configuration change to automated renewal contains a possible risk that the automated renewal process picks up some policies unacceptably late for renewal. Ultimately, the implementation of the `PolicyRenewalPlugin` plugin controls the lead time of any individual policy needs for renewal. By default, this process depends on the code of that plugin and the `NotificationConfig` system table accessed through the `NotificationConfigPlugin` plugin implementation. As you can overwrite the functionality of either plugin, which types of changes might significantly change the renewal start date can vary.

This process is not available from the PolicyCenter interface. It is also not possible to schedule this batch process. Instead, you must start this process using the `maintenance_tools -startprocess` command option with `PolicyRenewalClearCheckDate` or its equivalent web service command.

Run the Clear Policy Renewal Check Dates process

Run the Clear Policy Renewal Check Dates process to clear (null out) the existing dates for all policies through a single, direct, database update.

Before you begin

Only run this process if one or more of the following is true:

- If a configuration change to automated renewal contains a possible risk that the automated renewal process picks up some policies unacceptably late for renewal.
- After you bring the server back up after applying a configuration update.

Procedure

1. Place the server in `maintenance` mode (`NO_DAEMONS` or lower).
2. Run the following `maintenance_tools` command a single time only.

```
maintanance_tools -password password -startprocess PolicyRenewalClearCheckDate
```

3. After the process completes, finish bringing the server all the way up to production mode.

See also

- “Set the server run level through system tools” on page 87
- “maintenance_tools command” on page 426

Closed Policy Exception work queue

Code	`ClosedPolicyException`
Categories	`UIRunnable`, `Schedulable`
Implementation	Work queue
Schedule	Not scheduled
Class	`ClosedPolicyExceptionWorkQueue.java`

The Closed Policy Exception work queue runs policy exception rules on every closed `PolicyPeriod` that has not had exception rules run on it for a defined number of days.

Parameter `ClosedPolicyThresholdDays` in `config.xml` defines the minimum number of days that must pass before PolicyCenter runs the Exception rules again on a closed `PolicyPeriod`.

See also

- *Rules Guide*
- *Configuration Guide*

Create UW Rules for UW Issue Types work queue

Code	`CreateUnmappedUWRules`
Class	`CreateUnmappedUWRulesWorkQueue`
Categories	`UIRunnable`
Implementation	Work queue
Workers	4
Schedule	Not schedulable

After an upgrade, Create UW Rules for the UW IssueTypes work queue runs at server startup. If a `UWIssueType` object does not have a corresponding `UWRule` object, this process creates a `UWRule`, links the two objects, and marks the rule as externally managed.

Data Distribution batch process

Code	`DataDistribution`
Categories	`APIRunnable`
Implementation	Batch process
Stoppable	No (single phase process)
Schedule	Not schedulable
Class	`PCDataDistributionBatchProcess.java`

The Data Distribution batch process generates data on the distribution of various items in the PolicyCenter database. It is not possible to schedule this process. You must run this process from the **Data Distribution** screen of the Server Tools **Info Pages** or by using the `maintenance_tools` command line utility.

As this type of batch process can be very resource intensive, it has the possibility of adversely affecting the performance of the application. Before you run this process in a production environment, Guidewire recommends that you run the process first against a test environment that contains a full copy of production data.

See also

- "The Data Distribution screen" on page 374
- "maintenance_tools command" on page 426
- *Integration Guide*

Database Consistency Check work queue

Code	`DBConsistencyCheck`
Categories	`Schedulable`
Implementation	Work queue

Non-exclusive	Yes
Schedule	Not scheduled
Class	`DBConsistencyCheckWorkQueue.java`

The Database Consistency Check work queue runs consistency checks on the PolicyCenter database.

Use the Server Tools **Info Pages→Consistency Checks** screen to launch the checks from PolicyCenter. You can set the number and type of workers to use in running the consistency checks through this screen.

Alternatively, to schedule consistency checks, use the following `system_tools` command, adding the optional information on which checks to run against which tables:

```
system_tools -user user -password password -checkdbconsistency ...
```

See also

- “Work queues” on page 114 for a discussion of how PolicyCenter handles work queues.
- “Database consistency checks” on page 272 for an overview of database consistency checks
- “The Consistency Checks screen” on page 364 for details of the **Consistency Checks** screen in PolicyCenter
- “Command prompt tools” on page 421 for an explanation of command prompt options

Database Statistics work queue

Code	`DBStats`
Categories	`Schedulable`
Implementation	Work queue
Schedule	Not scheduled
Class	`DBStatisticsWorkItemWorkQueue.java`

The Database Statistics work queue generates database statistics about how the PolicyCenter application and data model interact with the physical database. For example, database statistics store row counts in a table, how a table distributes the data, and much more. A database management system uses statistics to determine query plans to optimize performance.

IMPORTANT Do not run or schedule this process if you set `<databasestatistics>` attribute `useoraclestatspreferences` to `true` in file `database-config.xml`.

Development mode

In development mode, it is possible to run the Database Statistics proess in any of the following ways:

- From a command prompt, using the `-updatestatistics` option of the `system_tools` command
- From the **Execution History** tab of the Server Tools **Database Statistics** screen
- As a scheduled batch process

Production mode

In production mode, it is possible to run Database Statistics process in the following ways only:

- From a command prompt, using the `-updatestatistics` option of the `system_tools` command.
- As a scheduled batch process

Guidewire recommendation

Guidewire specifically recommends that you collect full statistics in the following circumstances:

- If there are significant changes to data such as after a major upgrade.
- If using the `zone_import` command.
- If you are trying to troubleshoot performance problems.

In all other cases, Guidewire recommends that you collect INCREMENTAL database table statistics only.

See also

- "Understanding database statistics" on page 285
- "Managing database statistics using system tools" on page 287
- "The Database Statistics screen" on page 375
- "system_tools command" on page 429

Deferred Upgrade Tasks batch process

Code	`DeferredUpgradeTasks`
Categories	`APIRunnable`
Implementation	Batch process
Stoppable	No (single phase process)
Schedule	Not schedulable
Class	`DeferredUpgradeBatchProcess.java`

The Deferred Upgrade Tasks batch process creates the nonessential performance indexes and indexes on archived entities.

PolicyCenter runs the Deferred Upgrade Tasks process automatically after an upgrade if you set the following attribute on `<upgrade>` in `database-config.xml` to true:

```
defer-create-nonessential-indexes
```

If the `DeferredUpgradeTasks` process fails, manually run the process again during non-peak hours. To manually run the Deferred Upgrade Tasks process, use the following command:

```
maintenance_tools -server url -password password -startprocess DeferredUpgradeTasks
```

Note: To run this command, you must have permission to create indexes until after the `DeferredUpgradeTasks` batch process completes.

To check the status of the `DeferredUpgradeTasks` batch process, review the upgrade logs and the PolicyCenter Server Tools **Upgrade and Versions** screen.

Production mode

Do not go into full production while the Deferred Upgrade Tasks process is still running. The lack of so many performance-related indexes can likely make PolicyCenter unusable.

Until the Deferred Upgrade Tasks process has run to completion, PolicyCenter reports errors during schema validation while starting. These include errors for column-based indexes existing in the data model but not in the physical database and mismatches between the data model and system tables.

Do not use the PolicyCenter archiving feature until the Deferred Upgrade Tasks batch process completes successfully.

See also

- "The Upgrade and Versions screen" on page 397
- "maintenance_tools command options" on page 426
- *Upgrade Guide*

Destroy Contact for Personal Data work queue

Code	`DestroyContactForPersonalData`
Categories	`Schedulable`
Implementation	Work queue
Schedule	Not scheduled
Class	`PersonalDataContactDestructionWorkQueue.gs`

This work queue finds all `PersonalDataContactDestructionRequest` objects that have a status set to New or ReRun (category `ReadyToAttemptDestruction`). How far the destruction process went for the found contacts is determined by the `ContactDestructionStatus` returned from the Destroyer, the class that implements the PersonalDataDestroyer interface.

The contact destruction status is set to the returned status. If the status is Completed, Partial, or NotDestroyed (category `DestructionStatusFinished`), the date of completion is also populated.

An exception is thrown if return status is New or if you try to change the status from a typecode in the `DestructionStatusFinished` category.

See also

- *Configuration Guide*

Extract Rating Worksheets work queue

Code	`ExtractWorkSheets`
Categories	`UIRunnable`, `Schedulable`
Implementation	Work queue
Schedule	Not scheduled
Class	`WorksheetExtractWorkQueue.java`

The Extract Rating Worksheets work queue extracts rating worksheet data from worksheet container (`WorksheetContainer`) objects to files in a specified directory on the server with the `batch` role. The process also marks `WorksheetContainer` objects for purging.

This process calls the implementation of the `WorksheetExtractPlugin` plugin, which sets the target directory.

See also

- "Purge Rating Worksheets work queue" on page 152
- *Configuration Guide*

Form Text Data Delete batch process

Code	`FormTextDataDelete`
Categories	`UIRunnable`, `Schedulable`

Implementation	Batch process
Stoppable	No (single phase process)
Schedule	Once a day, at 6:00 a.m.
Class	`FormTextDataDeleteBatchProcess.java`

The Form Text Data Delete batch process deletes orphaned, purged, or archived `FormTextData` entities. A `FormTextData` object stores the text data that makes up the XML data for a `Form`.

Geocode Writer work queue

Code	`Geocode`
Categories	`UIRunnable`, `Schedulable`
Implementation	Work queue
Schedule	Not scheduled
Class	`GeocodeWorkQueue.java`

The Geocode Writer work queue is the writer for the Geocode process. This work queue runs periodically to update geocoding information on user contact (`UserContact`) primary addresses and account locations. The `UserContact` entity represents a PolicyCenter user.

See also

- "Understanding geocoding" on page 107
- "Configuring geocoding" on page 108

Group Exception work queue

Code	`GroupException`
Categories	`UIRunnable`, `Schedulable`
Implementation	Work queue
Schedule	Once a day, at 4:00 a.m.
Class	`GroupExceptionWorkQueue`

The Group Exception work queue runs any defined group exception business rules on all groups in the system.

See also

- *Rules Guide*

Handle Unresolved Contingency work queue

Code	`HandleUnresolvedContingency`
Class	`HandleUnresolvedContingencyWorkQueue.gs`
Categories	`UIRunnable`, `Schedulable`
Implementation	Work queue
Schedule	Every 2 hours

The Handle Unresolved Contingency work queue initiates action on pending contingencies with an action start date of today or in the past on which action has not yet started.

Adding contingency actions

You can also add contingency actions by modifying Gosu class `HandleUnresolvedContingencyWorkQueue`. Add a type code to the `ContingencyAction` type list. In `ContingencyEnhancement.gsx`, add an action start date for this action in the `updateActionStartDate` method. Add a property getter for the new job using `isPolicyChangeAction` as a model.

See also

- *Application Guide*

Impact Testing Export work queue

Code	`ImpactTestingExport`
Categories	`UIRunnable`
Implementation	Work queue
Schedule	Not schedulable
Class	`ImpactTestingExportWorkQueue.java`

The Impact Testing Export work queue exports test periods to an Excel file if you click **Create Excel Export File** on the **Impact Results** screen.

It is only possible to run the Impact Testing Export work queue from the Server Tools **Batch Process Info** screen.

See also

- *Application Guide*

Impact Testing Test Case Preparation work queue

Code	`ImpactTestingTestPrep`
Categories	`UIRunnable`
Implementation	Work queue
Schedule	Not schedulable
Class	`ImpactTestingTestPrepWorkQueue.java`

The Impact Testing Test Case Preparation work queue generates baseline policy periods on the selected policies if you click **Create Baselines** from the **Create Baseline** screen.

It is only possible to run the Impact Testing Test Case Preparation work queue from the Server Tools **Batch Process Info** screen.

See also

- *Application Guide*

Impact Testing Test Case Run work queue

Code	`ImpactTestingTestRun`

Categories	`UIRunnable`
Implementation	Work queue
Schedule	Not schedulable
Class	`ImpactTestingTestRunWorkQueue.java`

The Impact Testing Test Case Run work queue generates test policy periods rated using the selected rate books if you click **Quote Test Periods** from the **Testing Periods** screen.

It is only possible to run this process from the Server Tools **Batch Process Info** screen.

See also

- *Application Guide*

Job Expire work queue

Code	`JobExpire`
Categories	`UIRunnable`, `Schedulable`
Implementation	Work queue
Schedule	Once a day, at 6:00 a.m.
Class	`JobExpirationWorkQueue.java`

The Job Expire work queue expires a job if the job has had no action taken on it for a configurable period of time. This process changes the job status 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.

To configure the expiration threshold as the number of days past the effective date or the creation date, modify the corresponding configuration parameter in `config.xml`:

- `JobExpirationEffDateThreshold`
- `JobExpirationCreateDateThreshold`

Enable expiration for a specific job type

About this task

The Job Expire process examines all jobs meeting the date criteria. It then expires those jobs for which `job.canExpireJob` returns `true`.

Procedure

1. In the PolicyCenter Studio **Project** window, expand **configuration → config**:
   a. Open `config.xml`.
   b. Search for `JobExpireCheck<`*`JobType`*`>`, for example, `JobExpireCheckAudit`.
   c. Change the value from `false` to `true`.
2. (Audit only) In the **Project** window, expand **configuration→gwrc→gw→job**:
   a. Open `AuditProcess.gs`.
   b. Modify the `canExpireJob` method to return `true` instead of `false`.

      Often, business requirements do not permit the expiration of Audit jobs. To expire the audit job type, you need override the `AuditProcess.canExpireJob` method.

Result

Forcing the expiration of all jobs for which expiration is possible improves the performance of queries related to jobs.

See also

- *Application Guide*
- *Configuration Guide*

Notify External System for Personal Data work queue

Code	`NotifyExternalSystemForPersonalData`
Categories	`Schedulable`
Implementation	Work queue
Schedule	Not scheduled
Class	`PersonalDataDestructionNotifyExternalSystemsWorkQueue.gs`

This work queue finds all `PersonalDataDestructionRequest` objects that have a status typecode in the `DestructionStatusFinished` category and `RequestersNotified` set to `false`. Found requests are processed by sending a notification to all associated requesters, and `RequestersNotified` is then marked `true`. If the notification fails, an exception is thrown and `RequestersNotified` remains `false`.

> **Note:** The class that implements this work queue is `PersonalDataDestructionNotifyExternalSystemsWorkQueue`. In your implementation, you must verify that the notification was successful before marking `RequestersNotified` `true`.

A method on the `PersonalDataDestruction` plugin, `notifyExternalSystemsRequestProcessed`, is called by `PersonalDataDestructionNotifyExternalSystemsWorkQueue` to notify external systems when a personal data destruction request is completed. The original `RequestID` is passed to the method, which does nothing by default. You are expected to implement this method to notify systems of interest. The `RequestID` is received when the destruction request is originally created through the `PersonalDataDestructionAPI` web service.

> **Note:** In the base configuration, the class that implements the `PersonalDataDestruction` plugin is `PCPersonalDataDestructionPlugin`.

See also

- *Configuration Guide*

Open Policy Exception work queue

Code	`OpenPolicyException`
Categories	`UIRunnable`, `Schedulable`
Implementation	Work queue
Schedule	Not scheduled
Class	`OpenPolicyExceptionWorkQueue.java`

The Open Policy Exception work queue runs policy exception rules on every open, unlocked `PolicyPeriod` that has not had exception rules run on it for a definable number of days.

Parameter `OpenPolicyThresholdDays` in `config.xml` defines the minimum number of days that must pass before PolicyCenter runs the Exception rules again on an open and unlocked `PolicyPeriod`.

See also

- *Rules Guide*
- *Configuration Guide*

Overdue Premium Report work queue

Code	`OverduePremiumReport`
Categories	`UIRunnable`, `Schedulable`
Implementation	Work queue
Schedule	Once a day, at 4:30 a.m.
Class	`OverduePremiumReportWorkQueue.java`

The Overdue Premium Report work queue runs premium report escalation rules on overdue premium reports.

Phone Number Normalizer work queue

Code	`PhoneNumberNormalizer`
Categories	`UIRunnable`
Implementation	Work queue
Schedule	Once only, after an upgrade to 8.0.0+
Class	`CompactPhoneNormalizerWorkQueue.java`

IMPORTANT Run the Phone Number Normalizer work queue once only, after upgrading from earlier application versions to 8.0.0+. Disable the Phone Number Normalizer work queue in a production environment.

The Phone Number Normalizer work queue calls the registered plugin that implements the `IPhoneNumberNormalizer` interface. Use the Phone Number work queue to normalize phone numbers after upgrading from earlier versions of PolicyCenter to 8.0.0+.

Guidewire recommends that you use a substantial number of workers with the Phone Number Normalizer work queue. Using a small number of workers to normalize the phone numbers in a large database can take a very long time. The optimal number of workers to use varies according to the available hardware and the volume of the data involved. It is also possible to allocate workers to several different PolicyCenter servers rather then simply increasing the number of workers on a single server.

Disable this work queue after the process completes normalizing all old phone numbers by setting the number of workers in `work-queue.xml` to 0. You never need run Phone Number Normalizer work queue more than once, after an upgrade to 8.0.0+.

See also

- "Configuring work queues" on page 124
- *Upgrade Guide*

Policy Hold Job Evaluation work queue

Code	`PolicyHoldJobEval`
Categories	`UIRunnable`, `Schedulable`

Implementation	Work queue
Schedule	Once a day, at 2:00 a.m.
Class	`PolicyHoldJobEvalWorkQueue.java`

The Policy Hold Job Evaluation work queue evaluates each job against the policy holds blocking it. The process finds policy hold jobs that have the following characteristics:

- Jobs that are open
- Jobs that have policy periods with an active blocking hold
- Jobs that have not been evaluated since the time the policy hold changed

The `PolicyHoldJobEvalPlugin` plugin implementation determines the actions to take on each job found.

Policy Locations Risk Assessment Temporary Store work queue

Code	`PolicyLocationsRiskAssessment`
Class	`PolicyLocationsRiskAssessmentWorkQueue.gs`
Categories	`APIRunnable`
Implementation	Work queue
Schedule	Not scheduled

The Policy Locations Risk Assessment Temporary Store work queue starts risk assessments for all policy locations associated with a policy period. This work queue sends to the user requesting risk assessment an activity indicating whether the risk assessment succeeded or failed.

In the base configuration, PolicyCenter calls this process if you select **Update Risk Evaluations** on the **Risk Analysis** screen in a commercial property policy transaction.

This process retrieves risk assessment results from the Guidewire Spotlight risk assessment service for each location on a given `PolicyPeriod` and creates `LocationRiskAssessment` objects. Because PolicyCenter sets an effective date on each `LocationRiskAssessment` entity that it creates, it is only possible to run this process on an unlocked `PolicyPeriod` object.

If processing fails, the process does the following:

- It rolls back the entire transaction and does not persist any of the risk assessment results.
- It creates an activity and sends the activity to the list of configured user roles on the current job. You can configure the list of roles in the `SpotlightNotificationActivityCreator` class.
- It sends a notification activity to the requesting user, if the work item includes a reference to that user.

If processing succeeds, it does the following:

- It commits the transaction and persists the risk assessment results on the `PolicyPeriod`.
- It creates an activity and sends the activity to the list of configured user roles on the current job.
- It sends a notification activity to the requesting user, if the work item includes a reference to that user.

See also

- *Application Guide*

Policy Renewal Start work queue

Code	`PolicyRenewalStart`
Categories	`UIRunnable`, `Schedulable`
Implementation	Work queue

Schedule	Once a day, at 1:00 a.m.
Class	`PolicyRenewalStartWorkQueue.java`

The Policy Renewal Start work queue starts renewal processing for policies set to expire. The process determines the date on which to start the renewal process by subtracting the value of configuration parameter `RenewalProcessLeadTime` in `config.xml` from the policy expiration date.

Populate Search Columns batch process

Code	`PopulateSearchColumns`
Categories	`APIRunnable`
Implementation	Batch process
Stoppable	Yes (multi-phase process)
Schedule	Not schedulable
Class	`PopulateSearchColumnBatchProcess.java`

The Populate Search Columns batch process populates denormalized `searchColumn` columns from their designated `sourceColumn` columns.
This process is only available from the `maintenance_tools` command or from a web service.

See also

- *Configuration Guide*

Premium Ceding work queue

Code	`PremiumCeding`
Categories	`UIRunnable`, `Schedulable`
Implementation	Work queue
Schedule	Once a day, at 3:30 a.m.
Class	`RICedingWorkQueue`

The Premium Ceding work queue performs calculations related to premium ceding.

See also

- *Integration Guide*

Process Completion Monitor batch process

Code	`ProcessCompletionMonitor`
Categories	`UIRunnable`, `Schedulable`
Implementation	Batch process
Stoppable	No (single phase process)
Schedule	Not scheduled
Class	`ProcessCompletionMonitor.java`

The Process Completion Monitor batch process runs at schedulable intervals and examines the `ProcessHistory` table for all completed work queues and batch processes.

For each completed work queue that it finds, Process Completion Monitor:

- Determines if all the work items in that work queue have either completed or failed.
- Calls the `IBatchCompletedNotification` plugin implementation on a process if the process is complete and has no remaining available or checked-out work items.
- Sets `ProcessHistory.NOTIFICATIONSENT` to `true`, to prevent the process from invoking the `IBatchCompletedNotification` plugin implementation more than a single time for any given process.

The `IBatchCompletedNotification` interface has a `completed` method that you can override to perform specific actions after a work queue or batch process completes a batch of work.

See also

- "Performing custom actions after a process completes" on page 128.

Process History Purge batch process

Code	`ProcessHistoryPurge`
Categories	`UIRunnable`, `Schedulable`
Implementation	Batch process
Stoppable	No (single phase process)
Schedule	On the third day of the month, at 3:30 a.m.
Class	`ProcessHistoryPurge.gs`

The Process History Purge process purges batch process history data from the `ProcessHistory` table. It is important to periodically delete process history data. A large number of history records in the database can slow performance during use of the Server Tools **Batch Process Info** or **Work Queue Info** screens.

This process uses Gosu class `ProcessHistoryPurge` to read the value of the `BatchProcessHistoryPurgeDaysOld` parameter in `config.xml`. The process then uses this value to determine the history data to purge.

> **Note:** PolicyCenter also uses configuration parameter `BatchProcessHistoryPurgeDaysOld` to determine how many days to retain process history records, which the separate "Work Item Set Purge batch process" on page 157 process removes.

Product Model Pattern Activation batch process

Code	`ProductModelPatternActivation`
Categories	`APIRunnable`
Implementation	Batch Process
Schedule	Not scheduled
Class	`ProductModelPatternActivationProcess.java`

Product Model Pattern Activation batch process makes active newly defined product model patterns added during a rolling upgrade to a PolicyCenter server cluster. In a rolling upgrade, you bring down a single server at a time and perform a configuration upgrade on that server. After bringing that server back online, you do the same with the next server in the cluster and so on.

To provide for data integrity during this process, Guidewire disables any newly added product model patterns until all servers in the cluster have the same application configuration. Run the Product Model Pattern Activation process after you complete the rolling upgrade and after you flip the **Rolling Upgrade Complete** flag in the Server Tools **Upgrade and Versions** screen.

It is only possible to run this process using the following `maintenance_tools` command option:

```
maintenance_tools -password password -startprocess productmodelpatternactivation
```

See also

- "Updating product model patterns in a rolling upgrade" on page 201
- "Performing a rolling upgrade" on page 204
- "maintenance_tools command" on page 426

Purge work queue

Code	`Purge`
Categories	`UIRunnable`, `Schedulable`
Implementation	Work queue
Schedule	Every day at 4:30 a.m.
Class	`PurgeWorkQueue.java`

The Purge work queue purges jobs and prunes policy periods that meet the purge and prune criteria. This process deletes jobs and other entities from the database.

Note: Guidewire disables the Purge work queue in the base configuration.

See also

- To enable the Purge process, see the *Configuration Guide*.
- For information on the criteria that the batch process uses to determine eligibility for deletion from the database, see the *Configuration Guide*.

Purge Cluster Members batch process

Code	`PurgeClusterMembers`
Categories	`UIRunnable`, `Schedulable`
Implementation	Batch process
Stoppable	No (single phase process)
Schedule	On the first day of each month, at 2:00 a.m.
Class	`PurgeClusterMembers.gs`

The Purge Cluster Members batch process purges `ClusterMemberData` entities. This process uses Gosu class `PurgeClusterMembers` to read the value of the `ClusterMemberPurgeDaysOld` parameter in `config.xml`. The process then uses this value to determine to purge the `ClusterMemberData` entities that have a `LastUpdate` value prior to the current date minus the value of the `ClusterMemberPurgeDaysOld`.

Purge Failed Work Items batch process

Code	`PurgeFailedWorkItems`
Categories	`UIRunnable`, `Schedulable`
Implementation	Batch process

Stoppable	No (single phase process)
Schedule	On the first day of the month, at 1:00 a.m.
Class	`PurgeFailedWorkItems.gs`

The Purge Failed Work Items batch process purges failed work items from all work queues. This process uses Gosu class `PurgeFailedWorkItems` to determine which work items to delete.

During a scheduled execution of the batch process, the process deletes failed work items that are older than the last run date of the batch process. It then sets the last run date to the current date. Thus, if the scheduled execution of this batch process is monthly, the process deletes work items that are older than a month only.

If you run this process manually and there are work items that are newer than the last run date, the batch process does not delete them. If you then run the batch process a second time on the same day, the process deletes work items that are older than the current date. This is the expected behavior.

Purge Message History batch process

Code	`PurgeMessageHistory`
Categories	`UIRunnable`, `Schedulable`
Implementation	Batch process
Stoppable	No (single phase process)
Schedule	On the 20th of the month, at 1:00 a.m.
Class	`PurgeMessageHistory.gs`

The Purge Message History batch process purges old messages 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 the Purge Message History process removes the message.

Purge Old Transaction IDs batch process

Code	`PurgeTransactionIDs`
Categories	`UIRunnable`, `Schedulable`
Implementation	Batch process
Stoppable	No (single phase process)
Schedule	Not scheduled
Class	`PurgeTransactionIds.gs`

The Purge Old Transaction IDs batch process deletes SOAP header transaction IDs generated by systems external to PolicyCenter. This process uses Gosu class `PurgeTransactionIds` to read the value of the `TransactionIdPurgeDaysOld` parameter in `config.xml`. The process then purges transaction IDs that have a creation date prior to the current date minus the value of the `TransactionIdPurgeDaysOld` parameter.

Guidewire does not schedule this batch process in the base configuration as the table that stores the transaction IDs takes very little space in the database. Unless there is a constant buildup of these transaction IDs, there is no real need to continually purge this data. In fact, if you do purge this data, it is then not possible to determine if a new transaction is a duplicate of a transaction sent by the external system at an earlier date. There are other alternatives to purging this data. For example, you can partition the table by date.

See the PolicyCenter *Integration Guide* for information on SOAP headers. Also see `WsiCheckDuplicateExternalTransaction`.

Purge Orphaned Policy Periods work queue

Code	`PurgeOrphanedPolicyPeriods`
Categories	`UIRunnable`, `Schedulable`
Implementation	Work queue
Schedule	Not scheduled
Class	`PurgeOrphanedPolicyPeriodWorkQueue.java`

The Purge Orphaned Policy Periods work queue finds orphaned policy periods (policy periods not associated with a specific job) and deletes them. The process deletes policy periods and other entities from the PolicyCenter database. See "Purge Orphaned Policy Periods work queue" on page 151 for information on how PolicyCenter selects a policy period for deletion.

Note: Guidewire disables the Purge work queue in the base configuration. To enable this batch process, see the *Configuration Guide*.

See also

- *Configuration Guide*

Purge Profiler Data batch process

Code	`PurgeProfilerData`
Categories	`UIRunnable`, `Schedulable`
Implementation	Batch process
Stoppable	Yes (multi-phase process)
Schedule	Not scheduled
Class	`ProfilerDataPurgeBatchProcess.java`

The Purge Profiler Data batch process purges profiler data at regularly specified intervals. This process uses the read-only `ProfilerDataPurgeBatchProcess` class to read the value of the `ProfilerDataPurgeDaysOld` parameter in `config.xml`. The process then uses the value of this parameter to determine how many days to retain profiler data before Purge Profiler Data batch process removes it.

Purge Quote Clones work queue

Code	`PurgeQuoteClones`
Categories	`UIRunnable`, `Schedulable`
Implementation	Work queue
Schedule	Not scheduled
Class	`PurgeQuoteClonesWorkQueue.java`

Quote cloning creates copies of policy period quotes, known as quote clones. The Purge Quote Clones process deletes quote clones that are marked as processed from the PolicyCenter database.

By default, Guidewire disables this schedulable process in the base configuration.

See also

- *Application Guide*
- *Configuration Guide*

Purge Rate Book Export Result work queue

Code	`PurgeRateBookExportResult`
Class	`RateBookExportResultPurgeWorkQueue.gs`
Categories	`UIRunnable`, `Schedulable`
Implementation	Work queue
Schedule	Every day at 3 a.m.

The Purge Rate Book Export Result work queue removes spreadsheets and XML files resulting from exporting rate books to spreadsheet or to XML. This process removes `RateBookExportResult` Excel and XML files that are more than 60 days old. Specify the number of days in the `RateBookExportResultAgeForPurging` parameter in `config.xml`.

This process removes the files from the PolicyCenter database only. It does not remove files downloaded to the user's computer.

Disable this process by setting the `PurgeRateBookExportResultEnabled` parameter in `config.xml` to `false`.

See also

- *Application Guide*
- *Configuration Guide*

Purge Rating Worksheets work queue

Code	`PurgeWorksheets`
Categories	`UIRunnable`, `Schedulable`
Implementation	Work queue
Schedule	Not scheduled
Class	`WorksheetPurgeWorkQueue`

The Purge Rating Worksheets work queue removes worksheet container (`WorksheetContainer`) objects that meet certain criteria. The `RatingWorksheetContainerAgeForPurging` parameter in `config.xml` specifies the minimum number days after the closure of a job before the `WorksheetContainer` associated with its policy is eligible for purging.

See also

- "Extract Rating Worksheets work queue" on page 140
- *Configuration Guide*

Purge Risk Assessment Temporary Store work queue

Code	`PurgeRiskAssessmentTempStore`
Class	`PurgeRiskAssessmentTempStoreWorkQueue.gs`
Categories	`UIRunnable`, `Schedulable`

Implementation	Work queue
Schedule	Every Monday at 1:00 a.m.

The Purge Risk Assessment Temporary Store work queue deletes temporary objects created for risk assessment. This process deletes objects if they are older than the number of days specified in the `PurgeRiskAssessmentTempStoreDays` parameter. In the base configuration, Guidewire sets the value of this parameter to 30 days.

The Purge Risk Assessment Temporary Store process purges temporary objects if the course of standard PolicyCenter operations does not remove them.

See also

- *Configuration Guide*

Purge Temporary Policy Periods work queue

Code	`PurgeTemporaryPolicyPeriods`
Class	`PurgeTemporaryPolicyPeriodsWorkQueue.java`
Categories	`UIRunnable`, `Schedulable`
Implementation	Work queue
Schedule	Not scheduled

Sometimes policy period (`PolicyPeriod`) objects in a temporary (`Temporary`) status exist in the database. The Purge Temporary Policy Periods work queue removes these temporary policy periods from the database.

Configuration parameter `PurgeTemporaryPolicyPeriodsAfterDays` sets the minimum number of days that must pass before it is possible to purge a temporary policy period. In the base configuration, Guidewire set the value of this parameter to 14 day.

To enable this work queue, set configuration parameter `PurgeTemporaryPolicyPeriodsEnabled` to `true`. If set to `false`, the work queue does not remove any temporary policy periods.

Purge Workflow batch process

Code	`PurgeWorkflows`
Categories	`UIRunnable`, `Schedulable`
Implementation	Batch process
Stoppable	No (single phase process)
Schedule	On the first day of the month, at 1:30 a.m.
Class	`PurgeWorkflows.gs`

The Purge Workflow batch process purges completed workflows after resetting any referenced workflows. This process uses Gosu class `PurgeWorkflow` to read the value of the `WorkflowPurgeDaysOld` days parameter in `config.xml`. The process then uses this value to determine the number of days to retain workflow data before purging it.

Purge Workflow Logs batch process

Code	`PurgeWorkflowLogs`

Categories	`UIRunnable`, `Schedulable`
Implementation	Batch process
Stoppable	No (single phase process)
Schedule	On the first day of the month, at 2:30 a.m.
Class	`PurgeWorkflowLogs.gs`

The Purge Workflow Logs batch process purges completed workflows logs. This process uses Gosu class `PurgeWorkflowLogs` to read the value of `WorkflowLogPurgeDaysOld` parameter in `config.xml`. The process then uses this value to determine the number of days to retain workflow logs before purging them.

Rate Book Excel Export work queue

Code	`RateBookExcelExport`
Categories	`APIRunnable`
Implementation	Work queue
Schedule	Not scheduled
Class	`RateBookExcelExportWorkQueue`

The Rate Book Excel Export work queue exports a rate book and its included rate table and rate routines to a file in spreadsheet format. PolicyCenter initiates this process if a user selects **Export to Spreadsheet** in the **Rate Book** screen.

See also

- *Application Guide*
- *Configuration Guide*

Recalculate Contingency Action Start Date work queue

Code	`RecalculateContingencyActionStartDate`
Class	`RecalculateContingencyActionStartDateWorkQueue.gs`
Categories	`UIRunnable`, `Schedulable`
Implementation	Work queue
Schedule	Not scheduled

The Recalculate Contingency Action Start Date work queue recalculates the action start date on all unresolved contingencies. It is only necessary to run this process if you modify the `ContingencyEnhancement.gsx` code or want to update the action start date of existing contingencies.

See also

- *Application Guide*

Remove Old Contact Destruction Request work queue

Code	`RemoveOldContactDestructionRequest`
Categories	`Schedulable`

Implementation	Work queue
Schedule	Not scheduled
Class	`RemoveOldContactDestructionRequestWorkQueue.gs`

This work queue finds all `PersonalDataDestructionRequest`, `PersonalDataContactDestructionRequest`, and `PersonalDataDestructionRequester` objects that have the following values:

- `RequestersNotified` set to `true`
- `PersonalDataContactDestructionRequest.DestructionDate` plus the value of the `ContactDestructionRequestAgeForPurgingResults` configuration parameter is less than or equal to today's date

Each found request that has `AllRequestsFulfilled` equal to `true` is removed.

See also

- *Configuration Guide*

Reset Purge Status and Check Dates work queue

Code	`ResetPurgeStatusAndCheckDates`
Categories	`UIRunnable`
Implementation	Work queue
Schedule	Not schedulable
Class	`ResetPurgeStatusAndCheckDatesWorkQueue.java`

The Reset Purge Status and Check Dates work queue resets the purge status and purge or prune dates on the jobs. Run this process if you have a need to reset the purge status and purge date.

For each job, this 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 process resets the `PurgeStatus` to `Unknown`.

Retire Activities work queue

Code	`ActivityRetire`
Categories	`UIRunnable`, `Schedulable`
Implementation	Work queue
Schedule	Once a day, at 12:30 a.m.
Class	`RetireActivitiesWorkQueue.java`

The Retire Activities work queue retires `Activity` entity records that are either:

- Cancelled
- Dismissed

Retrieve Policy Terms work queue

Code	`RestorePolicyTerm`
Categories	`UIRunnable`, `Schedulable`
Implementation	Work queue
Schedule	Not scheduled
Class	`RestorePolicyTermWorkQueue`

The Retrieve Policy Terms work queue retrieves archived policy terms marked for retrieval. It is possible for a user to request the retrieval of an archived policy term. Retrieving a policy term generates an activity for the user who requested the retrieval of the policy term.

See also

- *Application Guide*

Solr Data Import batch process

Code	`SolrDataImport`
Categories	`UIRunnable`, `Schedulable`
Implementation	Batch process
Stoppable	No (single phase process)
Schedule	Not scheduled
Class	`SolrDataImportBatchProcess.gs`

The Solr Data Import batch process tests the operation of the free-text batch load command, especially its embedded SQL query. Only run the Solr Data Import process on development-mode servers.

IMPORTANT Do not run this process in production to load and re-index the Guidewire Solr Extension. Instead, run the free-text batch load command (`batchload`) on the host on which the Guidewire Solr Extension resides.

See also

- "Free-text batch load command" on page 317
- *Configuration Guide*

Team Screens batch process

Code	`TeamScreens`
Categories	`UIRunnable`, `Schedulable`
Implementation	batch process
Stoppable	No (single phase process)
Schedule	Every hour, at three minutes past the hour
Class	`TeamScreenProcess`

The Team Screens batch process collects statistics for the PolicyCenter **Team** tab screens.

See also

- *Application Guide*
- *Configuration Guide*

User Exception work queue

Code	`UserException`
Categories	`UIRunnable`, `Schedulable`
Implementation	Work queue
Schedule	Once a day, at 3:00 a.m.
Class	`UserExceptionWorkQueue.java`

The User Exception work queue runs the user exception rule set on all users in the system.

See also

- *Rules Guide*

Workflow work queue

Code	`Workflow`
Categories	`UIRunnable`, `Schedulable`
Implementation	Work queue
Schedule	Every 10 minutes
Class	`WorkflowDistributedWorkQueue.java`

The Workflow work queue wakes up at 10 minute intervals and runs workflow worker tasks. Workflow cannot advance any faster in the background than this schedule.

See also

- *Configuration Guide*

Work Item Set Purge batch process

Code	`WorkItemSetPurge`
Categories	`UIRunnable`, `Schedulable`
Implementation	Batch process
Stoppable	No (single phase process)
Schedule	On the second day of the month, at 1:30 a.m.
Class	`WorkItemSetPurge.gs`

The Work Item Set Purge batch process purges work item sets from the database. This process uses Gosu class `WorkItemSetPurge` to read the value of the `BatchProcessHistoryPurgeDaysOld` parameter in `config.xml`. The process then uses this value to determine the number of days to retain work item sets before purging them.

Note: The `BatchProcessHistoryPurgeDaysOld` parameter also configures how many days to retain process history records, which the separate "Process History Purge batch process" on page 148 process removes.

Work Queue Instrumentation Purge batch process

Code	`WorkQueueInstrumentationPurge`
Categories	`UIRunnable`, `Schedulable`
Implementation	Batch process
Stoppable	No (single phase process)
Schedule	On the second day of the month, at 2:30 a.m.
Class	`WorkQueueInstrumentationPurge.gs`

The Work Queue Instrumentation Purge batch process purges instrumentation data for work queues. This process uses Gosu class `WorkQueueInstrumentationPurge` to read the value of the `InstrumentedWorkerInfoPurgeDaysOld` parameter in `config.xml`. The process then uses this value to determine how long to retain work queue instrumentation data.

See also

- "The Work Queue Info screen" on page 356.

Unused and internal batch processes

PolicyCenter uses some processes either internally, or, in a few cases, not at all. You cannot run these processes from PolicyCenter, `maintenance_tools`, or a web services API.

Unused processes

The Batch Process Type typelist (`BatchProcessType.tti`) includes a few Guidewire platform processes that PolicyCenter does not currently use. Guidewire indicates this status by setting the `retired` flag on the process typecode to `true` and placing a line through the typecode the typelist. You can ignore these processes.

Internal processes

PolicyCenter uses some Guidewire batch processes internally. PolicyCenter runs these processes to generate database performance reports only. You cannot run these processes separately. They are:

- Microsoft DMV Report
- Oracle AWR Report

part 3

Server clustering administration

chapter 9

Understanding PolicyCenter server clustering

To improve performance and reliability, you can install multiple PolicyCenter servers in a configuration known as a cluster. A PolicyCenter 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 a PolicyCenter cluster functions.

See also

- *Installation Guide*

Cluster terminology

Guidewire uses the following terminology in discussions involving Guidewire PolicyCenter clusters.

Term	**Meaning**
Host	The physical machine on which one or more Guidewire applications run.
Application instance	An individual PolicyCenter deployment. It is possible to run multiple application instances on the same host.
PolicyCenter server	The server associated with each application instance. For production environments, Guidewire supports JBoss, Tomcat, WebLogic, and WebSphere servers. If running multiple servers on the same host, each server must map to a different physical port.
Server role	A categorization of each application instance in the cluster by its function, as defined by its role. Examples of server roles are `ui` (manages user interface requests), `batch` (manages batch processing), and `messaging` (manages messaging and message destinations). You define server roles using the cluster `<registry>` element in `config.xml`. See "Server roles" on page 169 for more information. See also "Understanding the configuration <registry> element" on page 60.
Cluster	A grouping of two or more Guidewire application instances that have a common configuration and function as an integrated unit. Typically, each server in the cluster has one or more roles or function. Most often, if there are multiple servers assigned the `ui` role, the cluster contains a third-party load balancer as well. The individual application instances in the cluster must all connect to a common database.
Cluster member	A single application instance within a Guidewire cluster.

Guidewire PolicyCenter cluster installations

The typical PolicyCenter cluster consists of two or more PolicyCenter servers, each of which has one or more server roles, and a load balancer. In general, a PolicyCenter cluster contains the following types of servers:

- One or more user interface servers (online servers) that process web requests, perform business transactions, and render web pages.
- One or more non-user interface servers (offline servers) that manage batch processing, work queues, scheduling, message destinations, and startable services (plugins).
- A load balancer that manages user interface requests if there are multiple user interface servers.

Technically, non-user interface servers can also process web requests from business users, meaning that they can also act as user interface servers. However, Guidewire recommends that you avoid redirecting business user web requests to non-user interface servers to simplify the management of the cluster.

Cluster membership

As a PolicyCenter server joins the cluster, it updates a membership table in the PolicyCenter database. All cluster servers periodically poll this table to determine cluster membership.

Cluster availability

To ensure a high degree of availability, Guidewire recommends that the cluster configuration include two or more servers with each specific server role. You also need to provide ample capacity for running role-constrained items such as message destinations or batch processes.

Cluster monitoring

Guidewire provides cluster monitoring screens that are available to those with privileges to view the Server Tools screens:

- The Server Tools **Cluster Members** screens provide information on each server in the cluster.
- The Server Tools **Cluster Components** screen provides information on the components running on a given server.

Also, there are `system_tools` command options that provide information on cluster members and components in the PolicyCenter cluster.

Cluster cache usage

Cluster inter-communication ensures that if an object changes in one cluster member cache, that member sends a cache invalidation message to the other members of the cluster. This message instructs the other cluster members to tag the cache entry for the object as obsolete and evict it from the cache. The next time a PolicyCenter server needs the object, it reloads the object value directly from the database. This mechanism is different from full cache synchronization, in which a server broadcasts the new value of the object to other cluster members.

It is possible to lose message packets. Network failures or other issues also can disrupt communication between cluster members. Such cases can result in cache eviction messages not being propagated to all cluster members. As a result, one or more cluster members can contain stale cache entries.

Guidewire applications implement a data versioning mechanism to prevent data corruption. One or more version mismatches indicates that one or more objects have changed since PolicyCenter last accessed the entities. This mismatch results in PolicyCenter issuing a Concurrent Data Change Exception (CDCE). The user or batch job can then re-issue a change based on the latest values entered.

See also

- "PolicyCenter server configuration" on page 59
- "Cache management" on page 94
- "Concurrent data change prevention" on page 95
- "Planning a PolicyCenter cluster" on page 177
- "Server roles" on page 169
- "Batch process prioritization" on page 194
- "Component load balancing" on page 190
- "The Cluster Members screen" on page 391
- "system_tools command options" on page 429

Cluster communication

In the base PolicyCenter configuration, PolicyCenter clusters use the following types of transport mechanisms for sending messages between cluster members:

- Reliable broadcast without replies
- Unreliable fast broadcast without replies
- Reliable unicast with reply

Guidewire provides a default plugin implementation to support each of these transport types. However, it is possible to implement your own unicast/multicast transport by implementing the corresponding plugin. Guidewire disables fast broadcast messaging in the base configuration.

Unicast communications

PolicyCenter clusters use PPP protocol over TCP for direct server-to-server communication. For example, it is possible for a PolicyCenter Server Tools screen function to create a message request that directly targets a specific server. In this case, server A, on which the message request originate, sends a unicast message to server B, who receives and processes the request. PolicyCenter server lease management also leverages unicast communication to speed up certain actions, such as lease transfers.

Multicast communications

PolicyCenter clusters leverage the database for distributing broadcast messages.

Cluster plugin implementations

In the base configuration, Guidewire provides the following plugin interfaces to support cluster communication:

`ClusterBroadcastTransportFactory`	Provides a single factory method for creating a cluster transport for reliable broadcast of messages, with no replies. PolicyCenter stores broadcast messages in the database and then periodically loads any new broadcast messages onto each node in the cluster. This type of cluster transport guarantees the delivery order and the reliable delivery of the broadcast message. PolicyCenter uses this mechanism for default message broadcast if you do not enable the `ClusterFastBroadcastTransportFactory` plugin implementation.
`ClusterFastBroadcastTransportFactory`	Provides a single factory method for creating a cluster transport for fast broadcast of messages, with no replies. This type of transport: • Uses UDP multicast protocol • Does not guarantee the delivery order or even the actual delivery of the broadcast message PolicyCenter typically uses this type of cluster transport for broadcasting cache eviction notifications to cluster members.

	The use of this transport type is optional. In the base configuration, Guidewire disables the `ClusterFastBroadcastTransportFactory` plugin implementation due to its use of the UDP protocol. If you do not enable the plugin implementation, PolicyCenter uses the `ClusterBroadcastTransportFactory` cluster transport for broadcast messages instead.
`ClusterUnicastTransportFactory`	Provides a single factory method for creating a cluster transport for point-to-point unicast messages between specific servers in the cluster. The default plugin implementation uses TCP for the transport protocol.

Guidewire provides internal Java classes for these cluster-related plugin implementations. It is not possible to modify these Java classes. To see the plugin definitions, open PolicyCenter Studio and navigate to the following location in the Studio **Project** window:

configuration→config→Plugins→registry

Configuring cluster communication

The cluster-related plugin implementations that Guidewire provides in the base PolicyCenter configuration are sufficient for most purposes. However, if you need more fine grained control over cluster communications, it is possible to use one of the following methods to provide that control:

- Plugin parameters
- System property overrides

For example, if you need precise control over binding ports, then use one of these methods to configure the ports directly.

Plugin parameters

The cluster plugin implementations that Guidewire provides in the base configuration all support plugin parameters that you can use to reconfigure the plugin. All plugin parameters are optional. Guidewire provides default values for each of the plugin parameters. See "Cluster plugin parameter reference" on page 164 for more information.

To define a plugin parameter, you manually add that parameter to the plugin definition in the PolicyCenter plugin editor. For example, suppose that you want to directly control the number of threads in the thread pool that handle inbound requests in the `ClusterUnicastTransportFactory` plugin. In this case, you manually add the `poolSize` parameter and value to the plugin definition for `ClusterUnicastTransportFactory`, using the Studio plugin editor.

System property overrides

PolicyCenter provides the ability to reconfigure your plugin configuration using system properties set from a command prompt as you start the application server. One advantage to using system property overrides to set transport values is that you do not have to modify the configuration inside a WAR/EAR file to do so. This makes it easier to use the same WAR/EAR file in different environments.

The exact syntax to use in setting system parameters at application server start is dependent on the application server type. See "Setting JVM options in PolicyCenter" on page 66 for more information. See "Cluster plugin system properties reference" on page 167 for a list of the system parameters that you can use with the clustering plugins.

Cluster plugin parameter reference

In the base configuration, PolicyCenter provides the following plugin implementations that manage message transport in a PolicyCenter cluster. Each of these plugin implementations provide a number of configuration parameters that you can set to precisely control such transport variables as server address and bind port number.

Plugin implementation	For more information
`ClusterBroadcastTransportFactory`	"Configuration parameters for ClusterBroadcastTransportFactory" on page 165
`ClusterFastBroadcastTransportFactory`	"Configuration parameters for ClusterFastBroadcastTransportFactory" on page 166

Plugin implementation	For more information
`ClusterUnicastTransportFactory`	“Configuration parameters for ClusterUnicastTransportFactory” on page 167

Defining a plugin parameter

To define a plugin parameter, you manually add that parameter to the plugin definition in the PolicyCenter plugin editor. For example, suppose that you want to directly control the number of threads in the thread pool that handle inbound requests in the `ClusterUnicastTransportFactory` plugin. In this case, you manually add the `poolSize` parameter and value to the plugin definition for `ClusterUnicastTransportFactory`, using the Studio plugin editor.

Configuration parameters for ClusterBroadcastTransportFactory

Use the configuration parameters in the following table to provide precise control over the plugin parameter values available in the implementation of `ClusterBroadcastTransportFactory`.

Parameter	Description
`batchesDeleteInterval`	Average time (in milliseconds) between the execution of a SQL statement that deletes old message batches from the database. Each server node in the cluster executes this SQL statement. Therefore, if the cluster installation contains many nodes, Guidewire recommends that you increase this value. The default is 60000 milliseconds (1 minute).
`batchKeepPeriod`	Maximum amount of time for PolicyCenter to retain a batch in the database before deleting it. The default is 600000 (10 minutes).
`batchReadInterval`	Maximum time interval (in milliseconds) between reading and receiving new batches. The default is 3000 milliseconds (3 seconds).
`batchWriteAttempts`	Maximum number of attempts to write to a batch queue. If the number of consecutive errors exceeds this threshold, the transport switches to ERROR mode in which each new messages pops the oldest message out of the in-memory queue. The purpose of this parameter value is to avoid out-of-memory issues. The default is 30.
`batchWriteInterval`	Maximum time interval to wait (in milliseconds) before PolicyCenter writes, or sends, the current batch of messages. The default is 2000 milliseconds (2 seconds).
`maxOutboundBufferSize`	Maximum size of outbound buffer (in megabytes). The purpose of this parameter value is to prevent out-of-memory issues if a transport is having problems writing or sending messages. The default is 25 megabytes.
`preferredBatchDataSize`	Maximum size of the batch (in bytes). If the size of the current batch (the sum of all of the message batch sizes) reaches this threshold, PolicyCenter writes, or sends, the current batch of messages immediately. This value must be less than or equal to the largest possible integer value supported by your hardware.
`preferrredBatchMessageCnt`	Maximum number of pending messages allowed in the batch queue. If the number of messages in the batch queue reaches this threshold, PolicyCenter writes, or sends, the current batch of messages immediately. The value must be less than or equal to the largest possible integer value supported by your hardware.
`receiverPoolSize`	Number of threads in the thread pool that handle inbound messages. The default is 4.

To set a plugin parameter, you must manually add that parameter to the plugin definition in the PolicyCenter plugin editor. To access the plugin editor, navigate to the following location in PolicyCenter Studio and double-click the plugin name:

configuration→config→Plugins→registry

See also

- *Configuration Guide*

Configuration parameters for ClusterFastBroadcastTransportFactory

Note: Guidewire disables the implementation of the plugin in the base PolicyCenter configuration. You must enable the plugin implementation before PolicyCenter recognizes any of these parameters.

Use the configuration parameters in the following table to provide precise control over the plugin parameters in `ClusterFastBroadcastTransportFactory`.

Parameter	Description
`bindAddr`	Inet address to which PolicyCenter is to bind. This parameter can be useful if there are multiple-NIC hosts. The default fallback for this parameter is the first non-loopback interface found on the host as defined by the `NetworkInterface` Java API.
`bindPort`	Port number to which PolicyCenter is to bind. This parameter can be useful for server hosts behind a firewall.
`maxMessageSize`	Maximum allowable size of message. PolicyCenter calculates the default value of this parameter using the following algorithm: (Maximum IP datagram size) - (UDP header size) - (IP header size) The maximum IP datagram size is 65,535. The UDP header size is 8. The IP header size is one of the following values: • IPv4 = 20 • IPv6 = 40 Thus, if using IPv6, the default value for this parameter is 65,535 - 8 - 40, which is 65,487.
`messageKeepPeriod`	Time (in milliseconds) to keep messages in memory in order to skip retransmitted messages and to combine divided messages. The default is one of the following: • 2 * (maximum retransmit interval) • 10 seconds, if not using retransmit
`messageSalt`	Integer value that PolicyCenter uses in calculating the sending message checksum. This value must be the same on all servers in the PolicyCenter cluster. The default is 12345.
`multicastAddress`	Multicast Inet address. The default is 228.8.8.8.
`multicastPort`	Multicast port. The default is 38180.
`nodeStatisticsKeepPeriod`	Time (in milliseconds) to keep node statistics in memory after last activity. The default is 3,600,000 (1 hour).
`oldMessagesDeleteInterval`	Average time between the removal old messages from the memory (in milliseconds). The default is 1,000.
`receiverPoolSize`	Number of threads in the thread pool that handle inbound messages. The default is 4.
`receiverQueueCapacity`	Thread pool queue capacity. The default is 100.
`retransmitIntervals`	Comma-separated list of retransmit intervals (in milliseconds). The default is 10000.
`sendHeartbeatInterval`	Time (in milliseconds) between sending heartbeat messages. The default is 30,000 (30 seconds).
`ttl`	Time-to-live (TTL) for multicast datagram packets. The default is 8.

To set a plugin parameter, you must manually add that parameter to the plugin definition in the PolicyCenter plugin editor. To access the plugin editor, navigate to the following location in PolicyCenter Studio and double-click the plugin name:

configuration→config→Plugins→registry

See also

- *Configuration Guide*

Configuration parameters for ClusterUnicastTransportFactory

Use the configuration parameters in the following table to provide precise control over the plugin parameters in `ClusterUnicastTransportFactory`.

Parameter	Description
`bindAddr`	Inet address to which PolicyCenter is to bind. This parameter can be useful if there are multiple-NIC hosts. The default fallback for this parameter is the first non-loopback interface found on the host as defined by the `NetworkInterface` Java API.
`bindPort`	Port number to which PolicyCenter is to bind. This parameter can be useful for server hosts behind a firewall. The default fallback for this parameter is an ephemeral port, a free port above 1024 that is within a range supplied by the host operating system.
`poolQueueCapacity`	Thread pool queue capacity. The default is 50.
`poolSize`	Number of threads in the thread pool that handle inbound requests. The default is 4.

To set a plugin parameter, you must manually add that parameter to the plugin definition in the PolicyCenter plugin editor. To access the plugin editor, navigate to the following location in PolicyCenter Studio and double-click the plugin name:

configuration→config→Plugins→registry

See also

- *Configuration Guide*

Cluster plugin system properties reference

You can use system properties, set at server startup from a command prompt, to modify certain parameters that affect message transport in the PolicyCenter cluster. Using these system properties, you can precisely control such transport variables as server address and bind port number to override the plugin parameter settings in the following plugin implementations:

- `ClusterFastBroadcastTransportFactory`
- `ClusterUnicastTransportFactory`

The following list describes the plugin parameters that you can set through system properties at application start.

Plugin type	Plugin parameter	System property
`ClusterFastBroadcastTransportFactory`	• `bindAddr`	• `gw.cluster.fudp.bind_addr`
`ClusterUnicastTransportFactory.bindAddr`	• `bindAddr` • `bindPort`	• `gw.cluster.nbtcp.bind_addr` • `gw.cluster.nbtcp.bind_port`

The exact syntax to use in setting system parameters at application server start is dependent on the application server type. See “Setting JVM options in PolicyCenter” on page 66 for more information.

Cache eviction messages

Guidewire does not guarantee the delivery of cache eviction messages in a PolicyCenter cluster. However, with that said, the following is true:

Message plugin type	Notes
`ClusterBroadcastTransportFactory`	It is unlikely for there to be a missed eviction message with this type of message transport as the message plugin reads the messages in order from the database. It is possible in extreme circumstances such as losing database connectivity for a time period longer than the configured cleaning interval for the broadcast table. However, it is not likely.
`ClusterFastBroadcastTransportFactory`	If enabled, PolicyCenter typically uses this type of cluster transport for broadcasting cache eviction notices to cluster members. As this type of message transport uses UDP packets, it is possible to drop a UDP packet during transmission. Is is also possible for PolicyCenter to miss a packet if a garbage collection operation takes longer than the wait time for retransmitting a packet. If dropped packets become an issue, try setting the `retransmitIntervals` parameter on the plugin to a smaller value than its 10 second default value.

Logging cluster plugin parameters

By default, PolicyCenter logs cluster-related information at the `INFO` level in the server log. The cluster information that PolicyCenter provides includes information on where the bind parameter values come, for example, from the bind property default or from a parameter definition in the plugin editor.

Plugin	Logging header
`ClusterFastBroadcastFactory`	`Server.Cluster.FastUdpMulticast`
`ClusterUnicastTransportFactory`	`Server.Cluster.PointToPoint`

Ephemeral port values

Unless you provide a value for the `bindPort` value on the `ClusterUnicastTransportFactory` plugin, PolicyCenter randomly selects a free port above 1024 that is within a range supplied by the host operating system. You can view information on an ephemeral port in the server log. Look for information that looks similar to the following:

```
Server.Cluster.PointToPoint Listener(host, port=#####): Listening
```

Cluster logging examples

In the first logging example, the cluster installation enables both the `ClusterFastBroadcastFactory` and `ClusterUnicastTransportFactory` plugins. The installation does not, however, provide separate values for the bind parameters for these plugins. Notice, therefore, that the following logging example provides information for both plugins and indicates that bind values are the default values (`first address of the first network interface`).

```
19:50:30,131 INFO Server.Cluster Starting cluster channel...
19:51:45,885 INFO Server.Cluster.FastUdpMulticast Bind address is /nn.nn.n.nnn
      (first address of the first network interface)
19:51:45,915 INFO Server.Cluster.PointToPoint com.guidewire.pl.cluster.internal.ptp
      .PointToPointUnicastTransport@xxxxxxxx: Starting...
19:51:48,047 INFO Server.Cluster.PointToPoint Bind address is /nn.nn.n.nnn
      (first address of the first network interface)
19:51:48,887 INFO Server.Cluster.PointToPoint Listener(serverid=prod1,port=53872): Listening...
19:51:48,954 INFO Server.Cluster.FastUdpMulticast Started on /nn.nn.n.nnn
19:51:48,959 INFO Server.Cluster Members joined the cluster: prod1 (xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx)
19:51:49,290 INFO Server.Cluster Inserted new record for prod1 node into cluster table
19:51:49,303 INFO Server.Cluster Cluster channel started.
```

In the second logging example, the cluster installation again enables both the `ClusterFastBroadcastFactory` and `ClusterUnicastTransportFactory` plugins. In this case, however, the installation provide a value for the `ClusterUnicastTransportFactory.bindPort` parameter, which is 53870, defined in the plugin editor for this plugin.

```
11:51:23,059 INFO Starting cluster channel...
11:51:23,250 INFO Bind address is /nn.n.n.n (first address of the first network interface)
11:51:23,273 INFO com.guidewire.pl.cluster.internal.ptp.PointToPointUnicastTransport@xxxxx: Starting...
11:51:23,375 INFO Bind address is /nn.n.n.n (first address of the first network interface)
11:51:23,375 INFO Bind address is 53870 (specified via plugin configuration)
11:51:23,377 INFO Listener(serverid=prod1,port=53870): Listening...
11:51:23,465 INFO Started on /nn.n.n.n
11:51:23,470 INFO Members joined the cluster: prod1 (xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx)
11:51:23,869 INFO Inserted new record for prod1 node into cluster table
11:51:23,877 INFO Cluster channel started.
```

Server roles

In general, Guidewire application cluster contains servers (cluster members) that manage the following types of functionality.

Function	Description
Online processing	Server interactively manages requests from users logged into Guidewire PolicyCenter.
Background processing	Server manages batch process execution, work queue processing, message destination processing, lease management, and other similar items.

Guidewire categorizes each individual server instance in the cluster by its function, as defined by its role. In the base configuration, PolicyCenter defines server roles to handle the following functionality. In a typical installation, only those servers that support external requests such as user input use the `ui` server role.

Server function	Server role	Servers with these roles manage...
Online processing	"ui server role" on page 172	• Interactions (user requests) between the PolicyCenter user interface and PolicyCenter itself. • Web service calls; there is no distinct server role for web services. However, any cluster member that receives a web service request can process that request by default. • Work queue processing, if there are distributed workers. Guidewire recommends that a server with the `ui` role handle these types of requests only during periods of low-load and during off-peak hours.
Background processing	"batch server role" on page 170	• Batch scheduling. • Batch process execution.
	"Messaging server role" on page 170	Message destination processing.
	"scheduler server role" on page 171	Schedule handling.
	"startable server role" on page 172	Startable service management.
	"workqueue server role" on page 172	Work queue management, if there are distributed workers.

It is possible for multiple servers in the PolicyCenter clusters to have the same server role. Servers that have the same role type typically have similar resource allocations and configuration. Conversely, servers with different server role types typically have different workloads and allocate their resources differently.

Server roles and lease managers

Guidewire associates a specific lease manager type with each server role. Thus, a message destination lease manager on a server with the `messaging` role manages message destinations for that server.

Each Guidewire server contains all types of lease managers. However, a lease manager only becomes active on a server with a server role that matches its lease type.

Server role validation

PolicyCenter performs the following server role validation at server start up:

- Validation of roles used in the application configuration files – If PolicyCenter detects the use of an unknown role in any of the configuration files, it throws an exception and refuses to start.
- Validation of roles assigned a specific server – If PolicyCenter detects the assignment of an unknown role to a server, it prints a warning to the server log and ignores the unknown role.

See also

- "Component lease management" on page 187
- "JVM options specific to the runServer build command" on page 66

batch server role

Guidewire PolicyCenter distributes batch processing across all server instances in the cluster that have the `batch` server role. At least one server in the PolicyCenter cluster must have the `batch` server role. It is possible to request the start of a batch process from any server in the cluster. However, only a `batch` server is capable of performing the work involved in the batch process. If the start request originates from a server that does not have the `batch` role, that server communicates the request to the other members of cluster. A server with the `batch` role accepts the request and performs the work.

Exclusive and non-exclusive batch processing

Guidewire categorizes batch processes into exclusive and non-exclusive batch processes:

- For exclusive batch processes, Guidewire guarantees that the batch process runs on exactly one cluster member with the `batch` role at a time.
- For non-exclusive batch processes, Guidewire guarantees that a single submission of a batch process runs exactly once. However, it is possible to submit non-exclusive batch processes for execution multiple times. In this case, it is possible for the batch process to run once for each submission, possibly concurrently, on multiple servers that have the correct server role.

Cluster members with the `batch` role use a batch-specific lease manager. Guidewire provides a configurable load balancing strategy for those servers with the `batch` role.

See also

- "Work queues and batch processes, a reference" on page 130
- "Understanding the configuration <registry> element" on page 60
- "Batch process prioritization" on page 194

Messaging server role

Message destinations and messaging server roles

In Guidewire PolicyCenter, it is possible to associate a specific server role with each message destination. In the base configuration, PolicyCenter does not specify a server role for any of the predefined message destinations. For all message destinations without a defined server role, PolicyCenter automatically sets the associated server role to the base configuration role of `messaging`.

The following table describes the relationship between server roles and message destinations.

Server	Messaging destination
Assigned `messaging` role	Can acquire the lease of any message destination
Assigned role or host name	Can only acquire the lease to a message destination with an assigned role that matches that of the server. For example, suppose that there is a message destination with an assigned role of `A`. Only a server that has role `A` can acquire the destination lease. A server with only role `B` cannot acquire the lease. It is possible to associate a specific host name with a specific message destination as well. Then, only that server host can acquire the lease to the message destination.
No assigned messaging role	Cannot acquire the lease of any message destination.

You associate a specific server role or host name with a message destination in the Guidewire Studio™ `message-config.xml` file. If you do not set a server role for a message destination, the Messaging editor shows a default role of `messaging` at the top of the screen.

You set a specific role on a PolicyCenter server in one of the following ways:

- Through an option on the `gwb runServer` command used to start the server from a command prompt.
- Through the `registry` metadata definitions in file `config.xml`.

Message destinations and leases

Guidewire PolicyCenter distributes message processing across all server instances that have the `messaging` server role (which is any role assigned to a message destination). Cluster members with these roles use a messaging-specific lease manager. In this case, a lease corresponds with a message destination, with each destination having a server declaration.

PolicyCenter creates message destination leases during messaging initialization. As each PolicyCenter starts, if it has a messaging role, it looks for a message destination to start. This message destination is any available message destination whose assigned role matches a role assigned to the server. PolicyCenter repeats this process until there are no more qualified destinations to assign.

Each messaging lease manager accepts a listener from the messaging system. Upon the activation of a lease operation by the lease manager such as deleting or expiring a messaging lease (and similar operations), the listener notifies the messaging system. PolicyCenter then stops messaging destinations that are deleted or acquired by other servers.

A shutdown command for a messaging destination initiates a request to expire the lease on the server for that destination. The destination listener accepts that request and passes the request to the lease manager for that destination.

PolicyCenter periodically reviews all destination leases to determine if the leases are still valid and to look for new leases to acquire. If a lease expires or the lease manager creates a new lease, PolicyCenter again searches for new messaging destinations to assign.

Guidewire provides a configurable load balancing strategy for those servers with the `messaging` role.

See also

- “Understanding the configuration <registry> element” on page 60
- “Component load balancing” on page 190
- “messaging_tools command” on page 427
- *Integration Guide*
- *Configuration Guide*

scheduler server role

Guidewire PolicyCenter typically utilizes only a small number of cluster members with the `scheduler` server role. These server instances run multiple synchronized instances of the scheduler in parallel.

IMPORTANT Each server with the `scheduler` role must also have configuration parameter `SchedulerEnabled` set to `true` in `config.xml`.

Servers with the scheduler role

Guidewire recommends that even small clusters have at least two servers with the `scheduler` role. This mitigates the possibility of a single server with the `scheduler` role becoming a single point of failure. However, Guidewire does not recommend more than four servers with this role in a cluster as large number of servers competing for database resources can possibly increase database contention.

See also

- "Understanding the configuration <registry> element" on page 60
- "The Startable Services screen" on page 391

startable server role

Guidewire PolicyCenter implements certain plugins (services) as cluster singletons. These plugins implement the `IStartablePlugin` interface and do not carry the `distributed` annotation. PolicyCenter runs a single instance of these plugins only, on a server with the `startable` server role. Cluster member with this role use a plugin-specific lease manager.

Guidewire provides a configurable load balancing strategy for those servers with the `startable` role.

Note: Guidewire defines an additional type of cluster singleton plugins, known as inbound integrations, in file `inbound-integration-config.xml`.

See also

- "Understanding the configuration <registry> element" on page 60
- "Component load balancing" on page 190
- "The Startable Services screen" on page 391
- *Integration Guide*

workqueue server role

Guidewire PolicyCenter distributes work queues across all server instances that have the `workqueue` server role. By default, each work queue starts a single worker on each server with the appropriate role, unless configured otherwise.

See also

- "Work queues" on page 114
- "Work queues and server roles" on page 127
- "Understanding the configuration <registry> element" on page 60

ui server role

Guidewire PolicyCenter uses the `ui` server role as a placeholder role only. Guidewire PolicyCenter servers typically operate in conjunction with a non-Guidewire load balancer that manages the user interface transactions. PolicyCenter distributes web requests to the various cluster members according to the rules specified for the load balancer. Any cluster server that receives a web request processes that request, regardless of role assignment.

See also

- "Understanding the configuration <registry> element" on page 60
- "Guidewire PolicyCenter cluster installations" on page 162
- "Planning a PolicyCenter cluster" on page 177

Example PolicyCenter cluster configuration

You assign a role to an individual server using the `<registry>` element in file `config.xml`. See "Understanding the configuration <registry> element" on page 60 for details.

The following sample code illustrates a PolicyCenter cluster definition with five cluster members.

```
<registry roles="batch, scheduler, workqueue, messaging, startable, ui>
  <server env="prod" serverid="pcnode1"roles="batch, scheduler, messaging, startable"/>
  <server env="prod" serverid="pcnode2"roles="batch, scheduler, messaging, startable"/>
  <server env="prod" serverid="pcnode3"roles="workqueue, ui" />
  <server env="prod" serverid="pcnode4"roles="workqueue, ui" />
  <server env="prod" serverid="pcnode5"roles="ui" />
</registry>
```

Cluster member startup

As each member of the cluster starts, a number of events occur:

- During startup, each PolicyCenter server runs a checksum on its own configuration. The server then compares its checksum with the configuration checksum stored in the database. If the startup server determines that the two configurations are not compatible, it does not start.
- As each PolicyCenter server joins the cluster, it updates a membership table in the PolicyCenter database. All cluster members periodically poll this table to determine cluster membership.
- During a rolling upgrade of the individual server members in the cluster, the startup server verifies that it is running either the source (original) or target (new) configuration. If the server detects any other configuration, it does not start.

See also

- "Guidewire PolicyCenter cluster installations" on page 162
- "Restarting the PolicyCenter cluster servers" on page 179
- "Supported rolling update changes" on page 199
- "Performing a rolling upgrade" on page 204

Cluster member shutdown

In general, a PolicyCenter cluster contains the following types of servers:

- One or more user interface servers that process web requests, perform business transactions, and render web pages.
- One or more non-user interface servers that manage batch processing, work queues, scheduling, message destinations, and startable services (plugins).

The shutdown procedure for a Guidewire PolicyCenter cluster member is dependent on the role of server involved.

See also

- "ui-role cluster member shutdown" on page 174
- "Non-ui role cluster member shutdown" on page 174

ui-role cluster member shutdown

Guidewire PolicyCenter provides the means to schedule a planned PolicyCenter server shutdown. You schedule a server shutdown through the Server Tools **Cluster Components** screen.

After you schedule a planned shutdown on a specific server:

- PolicyCenter shows an on-screen banner for that server instance that warns of the impending shutdown.
- The banner shows a count-down timer to the planned shutdown and recommends that any logged-in user log out of PolicyCenter.

See "Schedule a planned cluster member shutdown" on page 397 for more information.

Non-ui role cluster member shutdown

As with PolicyCenter user interface servers, you schedule a shutdown of a server cluster member with a non-user interface role through the Server Tools **Cluster Components** screen. However, there are additional requirements for shutting down a non-user interface server.

Stop any running background tasks in the following order:

Background task	Shutdown considerations
1. Scheduler	The scheduler typically runs on several servers, as a distributed component. Thus, stopping the scheduler on one server does not affect the cluster.
2. Batch processing	It is necessary for all running batch processes to complete before starting the server shutdown. In many cases, it is better to let a batch process complete without attempting to terminate the process. However, as some batch processes can take significant amount of time to complete, potentially hours, you often need to make an individual determination for each batch process. See "About planned server shutdowns" on page 392 for more information.
3. Work queue	It is necessary to stop all work queue workers on the server that you intend to shut down. As PolicyCenter distributes work queues across cluster members, stopping workers on one server does not typically affect the cluster. However, it is possible for the overall performance of work items processing to drop. Stop work queues after stopping batch processing to stop generating business transactions that can potentially generated messages.
4. Messaging destinations	It is necessary to stop messages destinations on the server that you intend to shut down. Other servers in the cluster must take over the stopped message destination.
5. Startable plugins	It is necessary to stop startable plugins (services) on the server that you intend to shut down. Other servers in the cluster must take over the stopped services.

Most background tasks, except batch processes, stop quickly as their units of work are small. The actual task managers, for example the Batch Process Manager of the Message Destination Manager, do not instantly stop in a server shutdown. Instead, each lease manager moves to a passive mode in which it does not start new background tasks and moves to stop or complete any currently running tasks.

After all components stop their background tasks, you can shut down the batch server safely.

Server shutdown monitoring

As it may take some time to stop all background tasks, you can use the following API method to track the background task shutdown progress:

```
ISystemTools.getClusterState()
```

You can also use the following `system_tools` command options to gather information about a server and the state of the server components:

```
system_tools -components
system_tools -nodes
```

See “system_tools command options” on page 429 for a discussion of these command options.

chapter 10

Working with a PolicyCenter cluster

This topic discusses ways to implement, manage, and monitor a Guidewire PolicyCenter cluster.

Planning a PolicyCenter cluster

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 a 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.

See also

- "Guidewire PolicyCenter cluster installations" on page 162
- *Installation Guide*

Create a Guidewire PolicyCenter cluster

To create a PolicyCenter cluster, first define the individual servers in configuration, then deploy your configuration to each server in the cluster.

Procedure

1. In your source configuration, for use on all PolicyCenter servers in the cluster, open `config.xml` for editing.
2. In `config.xml`, set the following clustering-related configuration parameters appropriately:

   ```
   ClusteringEnabled
   ClusterMemberPurgeDaysOld
   ConfigVerificationEnabled
   PDFMergeHandlerLicenseKey
   ```
3. Define the set of valid server roles for use in this cluster using the `<registry>` element in file `config.xml`.

Note: Guidewire recommends that you do not set the server roles on individual cluster members using the `<registry>` element in `config.xml`. Instead, use JVM system properties at server start to set the server roles for that server.

4. In `config.xml`, set the value for `KeyGeneratorRangeSize`.
5. Create a deployment WAR or EAR file for Guidewire PolicyCenter.
6. Install a PolicyCenter cluster server in the same way that you install a standalone PolicyCenter server.

   IMPORTANT If you install multiple PolicyCenter servers on the same host machine, each PolicyCenter server must run in its own JVM instance.

7. Start each member of the cluster in turn.

   As you start each server, set the server roles for that server using a JVM system parameter.
8. In PolicyCenter, navigate to the Server Tools **Cluster Members** screen and verify the information.

See also

- "Understanding the configuration <registry> element" on page 60
- "The Cluster Members screen" on page 391
- *Installation Guide*
- *Configuration Guide*

Managing a PolicyCenter cluster

This topic discusses the ongoing management of a clustered environment.

Enabling Guidewire clustering

To enable clustering in a Guidewire installation, set the `ClusteringEnabled` parameter in `config.xml` to `true` on all cluster members, for example:

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

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

See also

- *Configuration Guide*

Adding a server to a PolicyCenter cluster

PolicyCenter does not require that you use the cluster registry in file `config.xml` to define cluster members. Its use is optional in that regard. Instead:

- You can specify the server ID and server roles from the command prompt at server startup using JVM command options.
- Or, you can specify a subset of the cluster server members in the cluster registry and set other server properties from the command prompt.

For example, you can define the set of servers assigned the batch server role in the cluster registry and specify the remaining server IDs through JVM options at server start.

In adding a server to the PolicyCenter cluster:

Server not specified in `config.xml`

There is no issue. You can add a server to the cluster without any change of configuration or stopping and restarting servers.

Server specified in `config.xml`

It is possible to perform a rolling upgrade of each server instance in the cluster to upgrade the cluster registry of each individual server instance in turn.

See also

- "Understanding the configuration <registry> element" on page 60
- "JVM options and server properties" on page 64
- "Add a server to a PolicyCenter cluster" on page 179
- "Performing a rolling upgrade" on page 204
- "Supported rolling update changes" on page 199

Performing a cluster configuration upgrade

Guidewire provides a way to upgrade individual servers in a PolicyCenter cluster individually. This type of upgrade, called a rolling upgrade, is in contrast to a full database and application upgrade. Guidewire permits only a few specific changes to files, file types, and installation folders during a rolling upgrade of the individual servers in a PolicyCenter cluster.

See also

- "Supported rolling update changes" on page 199
- "Performing a rolling upgrade" on page 204

Add a server to a PolicyCenter cluster

Procedure

1. Deploy the current cluster configuration WAR or EAR file to the server that you intend to add to the PolicyCenter cluster.
2. Verify that this configuration has parameter `ClusteringEnabled` set to `true` in the configuration's `config.xml` file.
3. If using the `config.xml` server registry, you must set the `serverid` attribute for this server to a unique value within the cluster.

   If you use a non-unique `serverid` value, the server with the duplicate ID does not start and does not join the cluster. See "Understanding the PolicyCenter server environment" on page 60.
4. Start the new server.

Result

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

Restarting the PolicyCenter cluster servers

There are several different scenarios that require you to bring down one or more members of the PolicyCenter cluster and then restart the servers.

Server restart after server maintenance

In performing maintenance on the servers in the PolicyCenter cluster, do the following for each PolicyCenter server in the Guidewire cluster, as required:

- Set the server run level to `maintenance`.
- Perform the needed work.
- Restart the server.

See also

- "Place the server in maintenance mode" on page 89
- "system_tools command" on page 429

Server restart after rolling upgrade

In an application configuration upgrade (known as rolling upgrade), you bring down a single server, in the cluster, upgrade its application configuration, and bring that server back up. You then repeat this process with each member of the PolicyCenter cluster in turn, until all cluster members use the upgraded configuration.

IMPORTANT Start the configuration upgrade on a single cluster server and let it fully initialize before starting the upgrade process on the other cluster members.

Before starting a rolling upgrade, click **Start Rolling Upgrade** in the Server Tools **Upgrade and Versions** screen. You can do this on any server in the PolicyCenter cluster. This action indicates that a rolling upgrade of the individual cluster members is in progress.

After completing the upgrade of all cluster servers, click **Rolling Upgrade Complete** in the Server Tools **Upgrade and Versions** screen. This action indicates that all servers in the cluster now use the upgrade WAR/EAR file and that the rolling upgrade process is complete.

See also

- "Performing a rolling upgrade" on page 204
- "Unexpected upgrades" on page 208

Server restart after full PolicyCenter upgrade

In a full upgrade, you bring down all members of the PolicyCenter cluster completely. Before starting a full upgrade, click **Start Full Upgrade** in the Server Tools **Upgrade and Versions** screen. You can do this on any server in the PolicyCenter cluster. This action sets a flag in the PolicyCenter database to indicate that a full upgrade is in progress. As long as the upgrade-in-progress flag exists, it is not possible to start a PolicyCenter cluster member that uses the old (non-upgrade) WAR/EAR file.

In a full database upgrade, you must start a single cluster server and allow it to complete the full upgrade cycle before starting the upgrade on the remaining servers. After all the servers in the PolicyCenter cluster start with the upgrade WAR/EAR file, PolicyCenter automatically deletes the upgrade-in-progress flag from the PolicyCenter database.

Use the following guidelines as you bring up the individual cluster members:

- After the first cluster server completes the upgrade cycle, it is possible to bring up all other servers in the PolicyCenter cluster in parallel. However, if starting a large numbers of servers causes resource contention, insert a short interval of time between each server start, for example, 10 seconds.
- As a general rule, start servers that manage back-end processes first. For example, start servers with the `batch` and `messaging` roles before starting servers with the `ui` role.

See also

- *Upgrade Guide*

Server restart after database restore

It is sometimes the case that you want to restore the PolicyCenter cluster database from one of the following:

- A non-cluster server
- A server in a different cluster environment but with the same code base

Attempting to start the PolicyCenter server after the database restore can generates database errors sufficient to cause the server to not start. To prevent these errors, it is necessary to start a single cluster node and let it cycle through a full database upgrade before starting the other members of the PolicyCenter cluster.

To set the full upgrade flag on a server as you start the server, use the following JVM system property, with *`YYYYMMDD`* being today's date:

```
-Dgw.pc.full.upgrade.intended.date=YYYYMMDD
```

For example:

```
gwb runServer -Dgw.pc.full.upgrade.intended.date=20180726
```

After the first server completes the database upgrade operation, start the other servers in the PolicyCenter cluster.

See also

- “Unexpected upgrades” on page 208

Monitoring cluster health

You can monitor the health of a Guidewire application cluster in multiple ways.

Cluster member health

Typically, hardware or software load balancers check the health of the various cluster members and stop directing traffic to a cluster member that stops responding. This check is very summary and simply verifies that the corresponding network port responds. Therefore, it is possible that a load balancer redirects traffic to a cluster member 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

Some other infrastructure constraints exist. For example, some environments cannot use source IP stickiness to load-balance conversational SOAP calls. In this situation, if using Guidewire ContactManager, you can instantiate one ContactManager instance on each server machine hosting one or more PolicyCenter instances. You then configure each PolicyCenter instance on that machine to direct contact requests to that local ContactManager instance. In this scenario, if the local ContactManager instance is not functioning properly, it is advisable to stop directing traffic to any of the PolicyCenter instances on that server.

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
```

There are three possible responses by the 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 displays a specific ASCII character, depending on the circumstances.
- If the PolicyCenter server is not running, the browser displays an HTTP failure message, depending on the configuration of the server.

See the *Integration Guide* for a list of the specific ASCII characters that the ping utility can return. Invoking this utility programmatically provides more granular information on the server’s status.

It is possible to configure the load balancers access this URL on a regular basis to determine the health of each member of the cluster. You can then use these results to create redirection logic.

For example, suppose that you have an environment in which PolicyCenter directs traffic to a local ContactManager instance. You then configure the load balancer to only redirect traffic to a PolicyCenter instance if both that instance and the ContactManager instance on that server are accessible for user requests.

Cluster member monitoring in PolicyCenter

The Server Tools **Cluster** screens, accessible to system administrators, provide information on your PolicyCenter cluster installation. For PolicyCenter to make this screen visible, the value of cluster configuration parameter `ClusteringEnabled` in `config.xml` must be `true`.

The **Cluster** screens provide information on the server cluster installation that includes the following items.

Information about the local server member	Its server ID, the server roles assigned to this PolicyCenter server, and similar information
Information about individual members recognized by the cluster	Their server IDs, number of active user sessions on each server instance, the server roles assigned to each application instance, date and time of each server start, and similar information
Information on the components running on each cluster member	The component state, date and time of each component start, and similar information
Information on any component lease failover in progress	The date and time of the deadline in which to complete the failover process
History of each member in the cluster	The start and stop times for each cluster member, server roles, run level, and similar information

From this screen, on any cluster member, you can start or cancel a planned shutdown for any recognized server instance in the cluster.

See also

- See "The Cluster Members screen" on page 391 for more information on the Server Tools **Cluster** screens.
- See "Schedule a planned cluster member shutdown" on page 397 for information on starting or cancelling a planned cluster member shutdown.

Monitor cluster members using system tools

About this task

Guidewire provides several `system_tools` command options that provide information about individual members of the PolicyCenter cluster.

`-components`	Provides information about the components that exist on each PolicyCenter server in the cluster.
`-nodes`	Provides information about each PolicyCenter server in the cluster.

Procedure

1. Ensure that the PolicyCenter server is running.
2. Open a command prompt and navigate to the PolicyCenter installation directory:

```
admin/bin
```

3. Enter the following commands:

```
system_tools -user user -password password -components
system_tools -user user -password password -nodes
```

The PolicyCenter administration command prompt tools all require that you enter the password of an administrative user for the tool to work. The use of a user name is optional.

See also

- "system_tools command" on page 429

Using the PolicyCenter ping utility

Guidewire provides an unauthenticated web page that you can ping to access information about a PolicyCenter server. This is the second version of the ping utility, hence, `v=2` in the URL.

To access this web page, use the following URL:

```
http://server:port/pc/ping?v=2
```

The `ping` utility returns the following types of information on the return HTML page, depending on various factors, including whether the server is a production server and whether the server start was successful.

Information type	Result description
Planned shutdown status	Shutdown status (for example, planned, ready) if the server is involved in a planned shutdown.
Server run level code	ASCII decimal code. For example, 45, represents the '(' character.
Server run level name	Server run level, for example, `MULTIUSER`.
Server run level ordinal	QuickStart (Jetty) run level. For example, 5, corresponds to the `MULTIUSER` run level.
Server ID	Name of the server host.
Server up time	Time, in seconds, for which the server has been operational.
Server start up exception	Exception that caused the server to fail start up, if applicable. By default, the `ping` utility does not display this information on a production server.
Thread stack trace	Stack trace of the thread performing the transition from one server run level to another. By default, the `ping` utility does not display this information on a production server.

See also

- "Server run levels" on page 84

Using the ping utility with a production server

Due to security concerns, the PolicyCenter `ping` utility does not return certain information if used with a production server. In production mode, the `ping` utility does the following:

- It replaces the text of any exception that occurs at server start up with `<not null>`.
- It removes the stack trace of the thread that performs the work of transitioning from one server run level to another entirely.

To view this information for a production mode server, start the production server with the following JVM command option:

```
-Dgw.ping.servlet.show.stack=true
```

Example output from the ping utility

The following examples illustrate the kind of information that the server generates for the different server scenarios:

- Successful server start
- Failed server start
- Server transition between run levels
- Planned server shutdown

Successful server start

The following code is an example of the `ping` utility return values if the PolicyCenter server starts successfully.

```
{
  "runLevelCode": 50,
  "runLevelName": "MULTIUSER",
  "runLevelOrdinal": 5,
  "serverId": "PolicyCenterServer1",
  "uptimeSeconds": 45
}
```

Failed server start

The following code is an example of the `ping` utility return values if the PolicyCenter development server fails to start.

```
{
  "runLevelCode": 40,
  "runLevelName": "NODAEMONS",
  "runLevelOrdinal": 3,
  "serverId": "PolicyCenterServer1",
  "startupException": "java.lang.RuntimeException: Test Startup Exception\n\tat
        com.guidewire.pl.system.server.PingServerServletTest.
        testInitTabStateJSONObjectShowsStartupException(PingServerServletTest.java:52)\n
\tat... ",
  "uptimeSeconds": 40
}
```

The following code is an example of the `ping` utility return values if the PolicyCenter production server fails to start. By default, PolicyCenter does not show the actual exception text and instead replaces the text with `<not null>`.

```
{
  "runLevelCode": 40,
  "runLevelName": "NODAEMONS",
  "runLevelOrdinal": 3,
  "serverId": "PolicyCenterServerPROD1",
  "startupException": "<not null>",
  "uptimeSeconds": 40
}
```

Server transition between run levels

The following code is an example of the `ping` utility return values on a development server while it is transitioning from one run level to another. In this example, the server is transitioning from `MULTIUSER` run level to the `DAEMONS` run level.

```
{
  "attemptingTransition": {
    "fromRunLevelName": "MULTIUSER",
    "fromRunLevelOrdinal": 5,
    "threadStackTrace": "Thread-142:TIMED_WAITING\n\tat  sun.misc.Unsafe.park(Native Method)
\n\tat
          java.util.concurrent.locks.LockSupport.parkNanos(LockSupport.java:215)\n\tat...”,
    "toRunLevelName": "NODAEMONS",
    "toRunLevelOrdinal": 3
},
  "runLevelCode": 45,
  "runLevelName": "DAEMONS",
  "runLevelOrdinal": 4,
  "serverId": "PolicyCenterServer1",
  "uptimeSeconds": 6814
}
```

Note: Use `system_tools` command options to transition a PolicyCenter server from one run level to another.

Planned server shutdown

The following code is an example of the `ping` utility return values.

```
{
  "runLevelCode": 50,
  "runLevelName": "MULTIUSER",
  "runLevelOrdinal": 5,
  "serverId": "testsrv1",
  "uptimeSeconds": 1820
}
```

The following code is an example of the `ping` utility return values after starting a planned server shutdown from the (Server Tools) **Info Pages→Cluster Members** page.

```
{
  "plannedShutdownStatus": "activated",
  "runLevelCode": 50,
  "runLevelName": "MULTIUSER",
  "runLevelOrdinal": 5,
  "serverId":
  "testsrv1",
  "uptimeSeconds": 1825
}
```

The following code is an example of the `ping` utility return values after the server shutdown completes.

```
{
  "plannedShutdownStatus": "ready",
  "runLevelCode": 50,
  "runLevelName":  "MULTIUSER",
  "runLevelOrdinal": 5,
  "serverId": "testsrv1",
  "uptimeSeconds": 1830
}
```

See also

- “Using the ping utility with a production server” on page 183
- “Set the server run level through system tools” on page 87
- “system_tools command” on page 429

chapter 11

Working with component leases

This topic discusses server component lease managers, lease management, and component lease load balancing.

Component lease management

A lease represents the right to perform some job for some period of time. Within a PolicyCenter application cluster, a lease specifically represents one of the following:

- A single run of a batch process
- An instance of a message destination
- An instance of a startable plugin, if it is a single instance plugin

Each server instance in the PolicyCenter cluster has a lease manager for each lease type. However, some functionality require a server with a specific server role. For example, only a server with the `messaging` role can acquire and manage a lease for a message destination.

Each type of lease manager can perform the following actions:

- Create, acquire, renew, or retire a lease
- Release a lease or request the release of a lease from another lease manager
- Transfer a lease to, or request the transfer of a lease from, another lease manager

Lease management contention

PolicyCenter lease management provides for minimal contention if two lease managers attempt to acquire the same lease at approximately the same time. Guidewire guarantees that only one server can acquire a specific lease. Guidewire also guarantees that, for an exclusive batch process, only a single such lease can exist at any one time.

Simple lease management lifecycle for a batch process

The following is a general description of the lease management lifecycle for a batch process:

- PolicyCenter translates the request to execute a batch process into the creation of an available lease and broadcasts the lease availability to all servers in the cluster.
- A lease manager on a server with the batch role discovers the available lease and acquires it. No other server in the cluster can acquire the lease. The lease is exclusive.
- The server with the active lease executes the batch process until processing is complete, with the server extending the lease as needed.
- At the completion of the batch process, the server lease manager releases the lease.

Simple lease management lifecycle for a message destination

The following is a general description of the lease management lifecycle for a message destination:

- At message initialization, PolicyCenter creates each enabled destination as an available lease.
- A lease manager on a server with the `messaging` role discover the available leases and attempts to acquire a lease.
- As a server succeeds in acquiring a lease, it initializes the corresponding destination and places the destination into the proper state (started, stopped, or suspended).
- A destination runs as intended, based on its state.
- A server periodically extends its destination lease until the messaging daemon stops, such as during server shutdown or if you set the server run level to a level below DAEMONS.
- The server releases its destination lease.

Component lease failover

In the course of standard cluster operation, meaning no network issues, no lost cluster members or similar problems, PolicyCenter periodically and automatically renews all component leases. Thus, the expiration of a lease is an exceptional condition that requires attention. If a component lease expires, PolicyCenter considers the owner of that lease to have failed. To detect this situation, the cluster lease managers periodically search for expired leases and initiate failover of the expired lease to another member of the cluster.

After a lease manager detects an expired lease, it does the following:

- It terminates the lease.
- It updates the lease history.
- It deletes the lease or recreates the lease for acquisition by another cluster member, depending on the type of the lease.

Automatic and manual failover

Guidewire PolicyCenter supports both automatic and manual failover of a component lease from one cluster member to another member of the cluster. However, there are situations in which an automatic component failover is not desirable, for example:

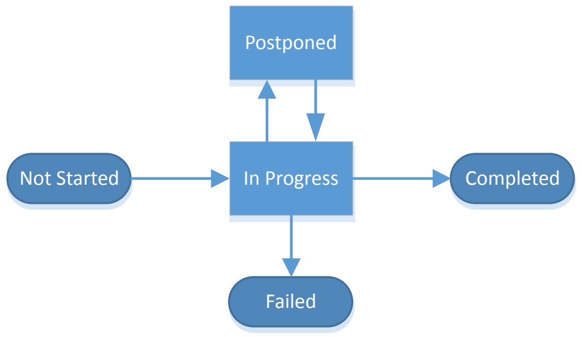
- There is a need for further additional configuration of an external third-party product before it is possible to start a destination or plugin on a different computer.
- There is a need for further diagnostic testing to determine the exact cause of failure before initiating the failover process.

See also

- See “Automatic failover of a component lease” on page 188 for a description of the automatic failover process that occurs after a lease expires.
- See “The background task failover plugin” on page 190 for a description of the default plugin implementation that Guidewire provides for handling component lease failover.

Automatic failover of a component lease

The following state diagram illustrates the different states in the automatic failover of a component lease. PolicyCenter defines these states in the `FailoverState` typelist. The `FailoverState` typelist does not contain the Completed state. The Completed state is implicit after the failover process completes and the lease manager deletes the original lease.



Initially, each component lease starts in the Not Started failover state. If a lease expires, the first lease manager that discovers the expired lease does the following:

- It sets the lease to the In Progress failover state. After set to this state, the component associated with the lease cannot run anywhere until there is a resolution of the issue that caused the lease to expire.
- It sets the **Retry Failover** field in the Server Tools **Cluster Components** screen to the following value.

```
CurrentTime + BackgroundTaskFailoverPlugin.FailoverTimeout
```

If more than one lease manager discovers the expired lease at the same time, only the first lease manager continues the failover handling. The other lease managers detect that their SQL updates do not change anything and do not continue the failover process for that lease.

The lease manager that started the failover calls the `handle`*`ComponentName`*`Failover` method on the `BackgroundTaskFailoverPlugin` plugin to determine what to do next with the lease. The method returns one of the following actions to handle the component lease failover.

Possible actions	Description
Complete the failover	The `BackgroundTaskFailoverPlugin` plugin logic confirms the lease failure and instructs the lease manager to complete the failover. In this case, the lease manager completes the failover process, either by deleting or expiring the lease.
Postpone the failover	It is possible that the `BackgroundTaskFailoverPlugin` plugin logic cannot reliably confirm the lease failure. In this case, it can postpone the failover process by returning an associated action to take and the time duration to wait before taking that action. The lease manager updates the **Retry Failover** field in the Server Tools **Cluster Components** screen with the following value: Current Time + `FailoverHandlingResult.Duration` After the updated retry failover time expires, the lease manager considers the lease expired and starts the process of lease failover again.
Dismiss the failover	It is possible that the `BackgroundTaskFailoverPlugin` plugin logic decides the specified background task did not fail, or, that this particular task requires some manual action. In this case, the `BackgroundTaskFailoverPlugin` plugin logic dismisses or fails the automatic failover of the lease. The lease with its `FailoverState` set to Failed remains in the database until there is some kind of manual intervention. The failover process does not attempt to retry the automatic failover.
Use external tool	The `BackgroundTaskFailoverPlugin` plugin logic returns a failover handled acton. This action instructs the lease manager to do nothing with the lease. An external tool either deletes or renews the lease. Calling an external tool to complete the failover can happen in any of the following ways: • Programmatically calling the `SystemToolsAPI.nodeFailed` method. • Programmatically calling the `SystemToolsAPI.completeFailedFailover` method. • Clicking the **Complete Failover** button on the Server Tools **Cluster Components** screen.

If the cluster member that started the failover does not complete the failover in the specified retry failover time, another cluster member detect this condition. The second cluster member then restarts the failover.

If at any point the original lease manager for the lease takes action to renew the lease, it does the following:

- It sets the `FailOver` state for the lease to Not Started.
- It resets the Retry Failover value to null.

At this point, the renewal of the lease resets the automatic failover process and negates any previous failover action undertaken for the renewed lease.

The background task failover plugin

In the base configuration, Guidewire provides a `BackgroundTaskFailoverPlugin` plugin implementation that you configure to manage component lease failover. The default implementation class for this plugin is Gosu class `DefaultBackgroundTaskFailoverPlugin`. This default implementation makes the implicit assumption that there is no manual cleanup work required for any batch process, message destination, or startable plugin after a component lease failover.

At the failover of a component lease, the failover logic postpones the start of the failover process by several minutes, the value of static variable `INITIAL_POSTPONE_TIMEOUT`. Failover logic implements the timeout to handle the case in which the database was unavailable for a relatively long period of time, and then comes back online. The failover postponement provides some time for the cluster members to recover and renew their leases.

After the failover postponement completes:

- If the cluster member that owns the lease does not return to the cluster, the automatic failover process continues.
- If the cluster member that owns the lease does return to the cluster, the automatic failover process fails.

Active external monitoring

Some clusters use external monitoring and management software to watch the JVM processes of cluster members. Guidewire provides Gosu class `ActiveExternalMonitoringBackgroundTaskFailoverPlugin` as a template that you can to use in such installations. You must implement your own logic for the `notifyExternalMonitoringAboutExpiredlease` method in this class.

Passive external monitoring

Guidewire provides Gosu class `PassiveExternalMonitoringBackgroundTaskFailoverPlugin` as a template to use in requesting a cluster member status report from external monitoring and management software. In this case, the external monitoring and management software is not actively watching the JVM processes of cluster members. Instead, it provides cluster member data on request.

Any monitoring and management software that you use for this purpose needs to be able to do the following:

- Check if a JVM process on a specified cluster member is alive and return the process uptime in seconds if the process is running.
- Terminate a specified JVM process.
- Start a new JVM process.

The plugin implementation manages the failover.

Component load balancing

In the base configuration, Guidewire provides several default strategies for managing the load balancing of messaging destinations, message processors, and startable services within the PolicyCenter cluster. Use plugin `BackgroundTaskLoadbalancingPlugin` to manage the load balancing process.

Load balancing strategies

In the base configuration, Guidewire provides the following strategies for load balancing components.

Work stealing	The work stealing strategy periodically transfers a component lease from an over-utilized server to a server that is under-utilized.

Work acquisition	The work acquisition strategy provides under-utilized servers with a chance to acquire a lease on an available component.

Plugin BackgroundTaskLoadBalancingPlugin

Guidewire provides the `BackgroundTaskLoadBalancingPlugin` plugin to manage the dynamic load balancing of long-running tasks such as startable services or message destinations. You access the plugin configuration in Studio at the following location:

configuration→config→Plugins→registry→BackgroundTaskLoadBalancingPlugin

In the base configuration, Guidewire provides the following default implementation class for the `BackgroundTaskLoadBalancingPlugin` plugin:

```
gw.api.system.cluster.DefaultBackgroundTaskLoadBalancingPlugin
```

This class implements the `BackgroundTaskLoadBalancingPlugin` interface. It is possible to modify the implementation of the default Gosu plugin class to meet your business needs.

In the base configuration, Guidewire sets the following Mode plugin parameters.

Name	Value
`startablePluginLoadBalancingMode`	`notransfer`
`messageDestinationLoadBalancingMode`	`notransfer`
`messageProcessorsLoadBalancingMode`	`dynamic`

Possible Mode values

Each of the Mode plugin parameters can take one of the following values.

Value	Description
`disabled`	Disables both the work acquisition and work stealing strategies.
`dynamic`	Enables both the work acquisition and work stealing strategies.
`notransfer`	Enables the work acquisition strategy only.

Methods on class DefaultBackgroundTaskLoadBalancingPlugin

Class `DefaultBackgroundTaskLoadBalancingPlugin` is the default implementation class for the `BackgroundTaskLoadBalancingPlugin` plugin. The `DefaultBackgroundTaskLoadBalancingPlugin` class implements the `BackgroundTaskLoadBalancingPlugin` interface, which contains the the following methods:

- `rebalanceStartablePlugins`
- `rebalanceMessageDestinations`
- `rebalanceMessageProcessors`
- `selectStartablePluginsToStartNow`
- `selectMessageDestinationsToStartNow`
- `selectMessageProcessorsToStartNow`

Method reference

`rebalanceStartablePlugins(LoadBalancingContext context)`

This method analyzes the startable plugin services that exist on the other member nodes of the PolicyCenter cluster and requests the transfer of those services to the current server, if the current server is underloaded.

This method returns the load balancing result as a `LoadBalancingResult` object. The result cannot be `null`.

`rebalanceMessageDestinations(LoadBalancingContext context)`

This method analyzes the message destinations that exist on the other member nodes of the PolicyCenter cluster and requests the transfer of those message destinations to the current server, if the current server is underloaded.

The method returns the load balancing result as a `LoadBalancingResult` object. The result cannot be `null`.

`rebalanceMessageProcessors(LoadBalancingContext context)`

This method analyzes the message processors that exist on the other member nodes of the PolicyCenter cluster and requests the transfer of those processors to the current server, if the current server is underloaded.

The method returns the load balancing result as a `LoadBalancingResult` object. The result cannot be `null`.

`selectStartablePluginsToStartNow(List<ComponentInfo> availableStartablePlugins, LoadBalancingContext context)`

This method selects the startable plugins to start. Use this method to set which components start on the current server and which components start on the other member nodes of the PolicyCenter cluster. The principal use of this method is to set a more uniform load distribution. If something delays the start of a component, and no other server starts it, PolicyCenter calls this method on the current server again.

This method returns the selected components to start as a list of `<ComponentInfo>` objects.

`selectMessageDestinationsToStartNow(List<ComponentInfo> availableMessageDestinations, LoadBalancingContext context)`

This method selects the message destinations to start. Use this method to set which components start on the current server and which components start on the other member nodes of the PolicyCenter cluster. The principal use of this method is to set a more uniform load distribution. If something delays the start of a component, and no other server starts it, PolicyCenter calls this method on the current server again.

This method returns the selected components to start, as a list of `<ComponentInfo>` objects.

`selectMessageProcessorsToStartNow(List<ComponentInfo> availableProcessors, LoadBalancingContext context)`

This method selects the message processors to start. Use this method to set which components start on the current server and which components start on the other member nodes of the PolicyCenter cluster. The principal use of this method is to set a more uniform load distribution. If something delays the start of a component, and no other server starts it, PolicyCenter calls this method on the current server again.

This method returns the selected components to start, as a list of `<ComponentInfo>` objects.

The LoadBalancingContext object

All `DefaultBackgroundTaskLoadBalancingPlugin` methods require that you pass a `LoadBalancingContext` object as you execute the method. Public interface `LoadBalancingContext` provides the following methods.

Method	Returns	Description
`getAllComponents()`	`List<ComponentInfo>`	Method returns a list of all components running in the cluster.
`requestTransferToThisServer(`*`component`*`)`	Boolean	Method requests the transfer of the given component to this server. Use the `getAllComponents` and the `ComponentInfo.getServerId` methods to obtain the current owner of a component. The component transfer itself is an asynchronous process. This method returns `true` if the transfer request completed successfully, which does not mean the transfer itself completed successfully.
`isAllowedToStartOnThisServer(`*`component`*`)`	Boolean	Method returns `true` if it is possible to start the given component on the current server.
`getTransferInfo(`*`component`*`, `*`unit`*`, `*`timeout`*`)`	`TransferInfo`	Given a specific component (*`component`*), the method returns information on the component transfer or `null` if the transfer attempt timed out.

Method	Returns	Description
		Parameters *timeout* and *unit* provide the time to wait before the transfer request times out.
`scheduleComponentAvailabilityCheck(`*`delay`*`, `*`unit`*`)`	--	Method schedules a component availability check. If a component is available to start, this method calls either of the following methods on the `BackgroundTaskLoadBalancingPlugin` plugin: • `selectMessageDestinationsToStartNow` • `selectStartablePluginsToStartNow` Parameters *delay* and *unit* provide the time to wait before performing the availability check.

The return `TransferInfo` object has two useful methods:

- `getCurrentOwner` - Method returns the current owner of the component. If the method returns `null`, it means that the component is already stopped.
- `getTargetOwner` - Method returns the owner of the target component. It cannot return `null`.

The ComponentInfo object

The `ComponentInfo` object contains information about a distributed component instance in the PolicyCenter cluster. Many of the `DefaultBackgroundTaskLoadBalancingPlugin` methods return a `ComponentInfo` object. For example, the `selectStartablePluginsToStartNow` method returns a `ComponentInfo` object for each startable plugin that the method starts. Public class `ComponentInfo` provides a number of getter and setter methods for the following component properties.

Property	Data type	Description
`type`	`ComponentType`	Type of the component, one of the following: • Startable service • Message destination • Message processor
`uniqueId`	`String`	UUID (universally unique ID) of the server on which the component is running.
`code`	`String`	Code designation of the component, `email` for a message destination, for example.
`name`	`String`	Name of the component, Email for a message destination, for example.
`state`	`ComponentState`	Assignment state of the component, for example: • Assigned • Unassigned • Ended
`serverId`	`String`	ID of the server that has the current lease on the component.
`startRequested`	`Date`	Date and time that the server received the component start request.
`started`	`Date`	Date and start time of the component.
`leaseExpiration`	`Date`	Date and time at which the component lease expires.
`transferRequested`	`Date`	Date and time at which the server received a request to transfer this component to another server.
`transferTarget`	`String`	The ID of the server to which the transfer request was made.
`terminateRequested`	`Date`	Date and time at which the server received a terminate request for this component.
`stopped`	`Date`	Date and time that the component stopped.
`retryFailover`	`Date`	Date and time of the deadline in which to complete the failover process.
`replacementId`	`String`	UUID of the server to which to transfer the component.

It is possible to see many of the values for these properties in the Server Tools **Cluster** screens, especially the **Components** screen.

See also

- *System Administration Guide*

The LoadBalancingResult object

A `LoadBalancingResult` object is the result of a load balancing operation. For example, the `rebalanceStartablePlugins` method on plugin `DefaultBackgroundTaskLoadBalancingPlugin` returns a `LoadBalancingResult` object. Public class `LoadBalancingResult` provides the following methods.

Method	Returns	Description
`getRepeatDelay()`	`long`	Method returns the delay value from load balancing result as a `long` integer.
`getRepeatUnit()`	`unit`	Method returns the delay time unit from load balancing result.
`isNeverRepeat()`	Boolean	Method returns one of the following: • `true` - The load balancing operation never repeats. • `false` - The load balancing operation repeats.
`repeatAfter(`*`unit`*`,` *`delay`*`)`	`LoadBalancingResult`	Static method that repeats the load balancing operation after the specified delay. Parameters *`delay`* and *`unit`* provide the time to wait before repeating the load balancing operation.. The *`delay`* value cannot be negative.
`neverRepeat()`	`LoadBalancingResult`	Static method that disable the load balancing operation after one attempt.

Batch process prioritization

In the base configuration, Guidewire provides an implementation of a lightweight strategy to prioritize the running of batch processes within the PolicyCenter cluster. File `batch-process-config.xml` provides a way to control the execution of multiple batch processes. This file has the following syntax:

```
<batch-process-config>

  <setting defaultServer="server_ID" env="environment" startupDelay="delay_list" startup-Timeout=n
        pollInterval=n/>
  <param name="parameter_name" value="parameter_value" env="environment"/>
  <batch-process typecode="type_code" env="environment" server="server_ID">
      <param name="parameter_name" value="parameter_value" env="environment"/>
  </batch-process>

</batch-process-config>
```

The following table describes the attributes on the `<settings>` element.

Element	Attribute	Required	Description
`settings`	`defaultServer`	Yes	Prefaced by the # (hashtag) symbol, this value specifies a server role. For example, `defaultServer="#batch"` means that every cluster member with the `batch` role is capable of running batch processes. Without the # symbol, this value specifies a server ID. For example, `defaultServer="node1"` means that the cluster member defined in the `<registry>` element in `config.xml` as `serverid=node1` is capable of running batch processes.
	`pollInterval`	Yes	Time, in seconds, between polling the database for new available leases.

Element	Attribute	Required	Description
			PolicyCenter also broadcasts information on new batch process leases. See “Simple lease management lifecycle for a batch process” on page 187 for more information.
	`startupDelay`	Yes	Number of seconds the batch process manager has to wait before starting the next process. The delay is dependent on the number of already running batch processes on the current server. In the base configuration, Guidewire sets this value to the following: `"0, 5, 15, 90, 180"` These values have the following meaning: 0 – Start the first process immediately 5 – Start the second process after 5 seconds 15 – Start the third process after 15 seconds 90 – Start the fourth process after 90 seconds 180 – Start the fifth and all subsequent processes after 180 seconds
	`startupTimeout`	Yes	Number of seconds allotted for a batch process to acquire a lease. If the time expires without the batch process acquiring the lease, PolicyCenter generates an error message similar to the following text (600 is the default value): `ERROR Server.BatchProcess No one started batch process after 600 seconds`
	`env`	No	Use to assign values for distinct installation environments. If used, this `<settings>` element applies only to the listed environments. To apply a specific `<settings>` element to multiple environments, input multiple comma-separated values for the `env` attribute. A multi-valued `env` attribute eliminates the need for a separate `<settings>` element for each environment.

The following table describes the attributes on the optional `<param>` element.

Element	Attribute	Required	Description
`param`	`name`	Yes	Name of the parameter. The only valid value is `RetryIfInitialConditionsFail`.
	`value`	Yes	Boolean value of `RetryIfInitialConditionsFail`.
	`env`	No	Use to assign a different value for `RetryIfInitialConditionsFail` to distinct installation environments or batch processes.

The following table describes the attributes on the `<batch-process>` element. The `<batch-process>` element also contains an optional `<param>` element that has the same meaning as listed in the previous table. If you set the `RetryIfInitialConditionsFail` parameter at the individual batch process level, it overrides any default value for this parameter set at the global level using the root-level `<param>` element.

Element	Attribute	Required	Description
`batch-process`	`typeCode`	Yes	Typecode value of a batch processes defined in the `BatchProcessType` typelist.
	`env`	No	Use to assign values for distinct installation environments. If used, this `<batch-process>` element applies only to the listed environments. To apply a specific `<batch-process>` element to multiple environments, you can input multiple comma-separated values for the `env` attribute. A multi-valued `env` attribute eliminates the need for a separate `<batch-process>` element for each environment.

Element	Attribute	Required	Description
	`server`	No	Specifies a server on which to run the batch process specified by the `typeCode` attribute.

The RetryIfInitialConditionsFail parameter

The optional `RetryIfInitialConditionsFail` parameter governs the behavior of a batch process depending on the return value of batch process class method `checkInitialConditions`. The name of the parameter is case-insensitive.

This parameter takes the following values.

`true`	If the batch process `checkInitialConditions` method returns `false`, the batch process does not terminate immediately. Instead, it continues to execute until its component lease expires. This is the batch process default behavior if the attribute is missing from `batch-process-config.xml`.
`false`	If the `checkInitialConditions` method returns `false`, the batch process terminates immediately.

The `<param>` element supports the `env` attribute, meaning that it is possible to set this parameter for different application environments, for example:

```
<paramname="RetryIfInitialConditionsFail" value="false"/>
<paramname="RetryIfInitialConditionsFail" value="false" env="dev"/>
<paramname="RetryIfInitialConditionsFail" value="true" env="prod"/>
```

Notice the following in this code example:

- The example code contains one parameter definition that has no defined `env` value set for the `RetryIfInitialConditionsFail` parameter.
- The example code does not contain a defined `RetryIfInitialConditionsFail` value for the `staging` environment

If a particular environment does not set the `RetryIfInitialConditionsFail` parameter, PolicyCenter does the following:

- It sets the value of the missing parameter for that environment to that of the parameter definition that does not have an `env` attribute set, if that parameter definition exists.
- If there is no parameter definition that does not have the `env` attribute set, PolicyCenter sets the value of the missing parameter to the `RetryIfInitialConditionsFail` default value, which is `true`.

It is also possible to set this parameter individually on each PolicyCenter batch process by including the optional `<param>` element on the `<batch-process>` element definition. If you set the `RetryIfInitialConditionsFail` parameter at the individual batch process level, it overrides any default value for this parameter set at the global level using the root-level `<param>` element.

chapter 12

Deploying supported changes in a clustered environment

Guidewire provides a way to deploy supported changes to individual production servers in a PolicyCenter cluster. Guidewire calls this type of deployment a rolling upgrade, in the sense that upgrade changes move through the cluster, one server instance at a time. This type of deployment is in contrast to a full database and application upgrade. A full upgrade requires that you bring down all PolicyCenter production servers in the cluster to complete the upgrade. A rolling upgrade allows you make certain changes on a server by server basis without bringing down the entire production cluster to make the changes.

Rolling upgrade overview

To support the need to perform regular updates without stopping the entire production installation, Guidewire provides a way to update each application instance in a PolicyCenter production cluster separately. Guidewire calls this type of upgrade a rolling upgrade. In a rolling upgrade, it is possible to stop an individual PolicyCenter server and deploy supported changes to that server without stopping the other servers in the cluster. In this way, PolicyCenter continues to function even through the upgrade process.

For this to be possible, Guidewire restricts the types of changes that are possible with a rolling upgrade. The target (new) application build must be compatible with the source (old) application build. Guidewire provides a command prompt tool that you use to determine if the two application builds are compatible. If the two builds are compatible, you can perform a rolling upgrade of each server instance in the cluster. If the two builds are not compatible, you must perform a full application and database upgrade.

The requirements for a rolling upgrade are much simpler than for a full application upgrade. Thus, it is possible for a PolicyCenter system administrator to perform a rolling upgrade of the cluster server members. Alternatively, a member of your IT department can perform this type of upgrade.

IMPORTANT You cannot use a rolling upgrade to upgrade from a major or minor version of Guidewire PolicyCenter to another major or minor version of PolicyCenter. In almost every case, a rolling upgrade is not suitable for Guidewire application patches or maintenance releases. Only if an application patch meets the compatibility criteria necessary for a rolling upgrade is a rolling upgrade of that patch possible. A rolling upgrade is not a replacement or substitute for a full application and database upgrade.

Rolling upgrade and archiving

It is not possible to perform any kind of archiving operation, either archive or restore, while a rolling upgrade is in progress. PolicyCenter throws an exception on any attempt to do so.

Risks associated with rolling upgrade of cluster server members

In performing a rolling upgrade, you trade safety for convenience. In particular, there are risks associated with modifying any configuration related to distributed processing such as batch processes and work queues. Do not attempt to perform a rolling upgrade of the production PolicyCenter cluster if your changes affect those areas of the application.

It is possible for individual PolicyCenter servers in the cluster to fail the rolling upgrade process for a variety of reasons. If so, you need to spend time testing and recovering from a failed attempt to return a server to a safe state. In particular, you need to review any entities that possibly changed due to the deployment process.

Assumptions around a rolling upgrade

As a rolling upgrade occurs in a live production environment, it is important to understand the assumptions underlying a rolling upgrade.

After starting the rolling upgrade:

- Guidewire expects most PolicyCenter users to continue to interact with the cluster members running the source build.
- Guidewire expects only a small set of users to interact with the cluster members running the target build.
- Guidewire expects a user to not switch between the source and target configurations as a matter of course. For example, Guidewire does not expect a user to process the same policy on two different cluster members with different configurations by switching back and forth between the configurations.

Guidewire expects you to execute and complete a rolling upgrade during a single scheduled maintenance window. The maintenance window is a period of low activity that spans a duration of hours, not days.

See also

- “Supported rolling update changes” on page 199
- “Verification of upgrade compatibility” on page 204
- “Unexpected upgrades” on page 208

Differences between a rolling upgrade and a full upgrade

Guidewire provides two different ways to modify the configuration of an application server in a PolicyCenter production cluster:

- Full upgrade
- Rolling upgrade

Each of these upgrade types has associated advantages and disadvantages. You use each upgrade type in different circumstances.

Full application and database upgrade

A full application upgrade has the following functionality:

- Supports either a single or multiple server configuration and upgrade.
- Supports making significant changes to both the PolicyCenter application and database.
- Requires that you stop the entire PolicyCenter production installation for a period of time.
- Requires a significant amount of testing before and after complex changes.

It is possible to start a full upgrade from any server instance in the PolicyCenter cluster, from the Server Tools **Upgrade and Versions** screen.

Rolling cluster member upgrade

A rolling upgrade of the individual server members in the cluster has the following functionality:

- Supports configuration deployment to multiple server instances.
- Successively targets a single member of a production cluster for configuration deployment while leaving the remaining cluster members available for the processing of PolicyCenter operations.
- Supports a limited set of configuration changes.
- Requires that the new target configuration be compatible with the existing source configuration.
- Supports a command-line utility that checks the compatibility of the source and target configurations before deployment.
- Supports configuration deployment management and tracking from within PolicyCenter.

It is possible to start a rolling upgrade from any server instance in the PolicyCenter cluster, from the Server Tools **Upgrade and Versions** screen.

See also

- "Supported rolling update changes" on page 199
- "Verification of upgrade compatibility" on page 204
- "The Upgrade and Versions screen" on page 397

Supported rolling update changes

Guidewire permits changes to the following files, file types, and PolicyCenter installation folders during a rolling upgrade of the individual members of a cluster.

File or folder	Safe actions
`config.xml`	It is safe to change the value of a configuration parameter that Guidewire designates as possible to vary on different servers in the same environment. Guidewire lists these parameters as *Set for Environment: Yes* in the parameter description. To access a list of configuration parameters for review, see .
`database-config.xml`	It is safe to change the value of attribute `useoraclestatspreferences` (from `false` to `true` or `true` to `false`) during a rolling upgrade. However, if you reset the value of this attribute from `true` to `false`, you must also perform manual steps to complete the process. See "Revert to DBStats batch processing for database statistics" on page 293 for more information.
`gsrc`	It is generally safe to add new Gosu classes during a rolling upgrade. However, it is unsafe to modify or remove a Gosu class in a rolling upgrade. See "Making changes to Gosu code in a rolling upgrade" on page 200 for more information.
`import`	It is safe for the `import` directory to differ across individual PolicyCenter instances, as long as the database is not empty on server startup. This is because the `import` directory contains files that PolicyCenter imports by default on server startup into an empty database.
`Localizations`	It is safe to add, delete, or modify any of the display keys in the various `display_`*`LanguageCode`*`.properties` files.

File or folder	Safe actions
	IMPORTANT It is not safe to make changes to either `language.xml` or `localization_LocaleCode.xml` as these files must remain consistent across all cluster members.
`plugin`	It is safe to modify a `.gwp` file in the root `plugin` directory, including pointing to a new implementation class. Guidewire does not permit the following with respect to plugins in a rolling upgrade: • Modifications to non-distributed, startable plugins • Modifications to files in the `plugins/Gosu` or `plugins/shared` directories Changes to plugin implementations can cause individual PolicyCenter instances to have different versions of an object. Thus, it is possible that PolicyCenter instances running the source configuration to consider objects create or updated on the target configuration to be invalid.
`resources/productmodel`	It is only safe to make changes to specific product model pattern types, as well as to very narrow changes to those types. See "Updating product model patterns in a rolling upgrade" on page 201 for more information.
`rules`	It is safe to perform the following operations on PolicyCenter Gosu rules: • Add a new rule • Modify an existing rule, including enabling or disabling the rule • Rename a rule, which is actually deleting a rule and adding the rule under a different name Changes to validation rules can cause individual PolicyCenter instances to have different versions of an object. Thus, it is possible that PolicyCenter instances running the source configuration. to consider the objects created or updated on the target configuration to be invalid.
`servlet`	It is generally safe to modify existing Gosu servlet implementations in a rolling upgrade. However, it is not permissible to modify `config/servlet/servlets.xml` in a rolling upgrade. **IMPORTANT** Guidewire recommends that you undertake thorough testing after making a change to a Gosu servlet implementation to verify that all product integrations continue to work as intended.
`templates`	It is safe to make modifications to note, email, and document templates in the following directories as the impact of a change to a template affects only that template. • `config/resources/doctemplates` • `config/resources/emailtemplates` • `config/resources/notetemplates`
`typelists`	It is generally safe to add typecodes to an existing typelist or to add an entirely new typelist. Also, it is safe to edit the typelist description or change its category. **Note:** If you add a typelist, the typelist shows on servers running the target (new) configuration only. On servers running the source (old) configuration, the typelist shows as blank. See "Making changes to typelists in a rolling upgrade" on page 201.
`web`	It is safe to add or a delete a PCF or to modify an existing PCF.
`webservices`	It is generally safe to make changes to web services. However, a change to an existing web service has the potential to break integration with a third-party product. **IMPORTANT** Guidewire recommends that you undertake thorough testing after making changes to web services configuration to verify that all product integrations continue to work as intended.

See also

- "Verification of upgrade compatibility" on page 204

Making changes to Gosu code in a rolling upgrade

It is generally safe to add new Gosu classes to Guidewire PolicyCenter during a rolling upgrade. However, as Gosu is prevalent throughout PolicyCenter, modifications to Gosu code during a rolling upgrade can affect nearly every part of the behavior of PolicyCenter. Thus, Guidewire explicitly recommends that you do not modify existing Gosu classes during a rolling upgrade.

By definition, a rolling upgrade means that some nodes in a multi-node system pick up changes from the upgrade before other nodes. Thus, changes in Gosu code can cause some nodes to behave differently than other nodes while the rolling upgrade is in progress. For a full upgrade, this is not an issue, because you must shut down every node in the cluster before you start the upgrade. However, for a rolling upgrade, the lack of a consistent behavior is a major concern.

Do not modify, change, or delete Gosu code during a rolling upgrade of a PolicyCenter cluster.

Making changes to typelists in a rolling upgrade

Guidewire permits only certain changes to typelists during a rolling upgrade of application servers in a PolicyCenter cluster.

Typelist changes that are safe to make in a rolling upgrade

In general, the following changes to typelists are safe to make in a rolling upgrade:

- Adding a new typelist
- Adding one or more typecodes to an existing typelist, either by extension or otherwise
- Modifying the name, description, priority, and sort order of an existing typecode

Typelist changes that are not safe to make in a rolling upgrade

It general, the following changes to typelists are not safe to make in a rolling upgrade:

- Deleting an existing typecode
- Renaming an existing typecode, which is essentially a delete and add operation
- Adding a typecode to a typelist marked as `final`

Typelists that you cannot change in a rolling upgrade

There are specific typelists for which it is not safe to make a change of any kind in a rolling upgrade. These typelists are:

- `BatchProcessType`
- `GroupType`
- `LanguageType`
- `SystemPermissionType`
- `UserRole`
- `ValidationLevel`

Guidewire maintains the list of blacklisted typelists in the following internal file:

- `pl-rolling-upgrade-typelist-blacklist.txt`

It is not possible to modify the list of blacklisted typelists in any way.

Updating product model patterns in a rolling upgrade

Guidewire does not support all product model pattern changes during a rolling upgrade of the individual members of a PolicyCenter server cluster. In general, Guidewire only supports the addition of new product model patterns for types that support availability logic.

It is possible to make narrowly specific changes safely to the following types of product model patterns in a rolling upgrade:

- “Coverage, condition and exclusion patterns” on page 202
- “Offering patterns” on page 202
- “Question set and question patterns” on page 202
- “Modifier patterns” on page 203
- “Lookup, grandfathering, and modifier minmax data” on page 203

Guidewire only supports changes to types of product model patterns and other product model information that the above list explicitly describes.

Important

1. PolicyCenter provides the ability to reload product model availability data through the Server Tools **Product Model Info** screen, by clicking **Reload Availability**. Guidewire disable this button during a rolling upgrade. The

button does not become active until a user clicks **Rolling Upgrade Complete** in the Server Tools **Upgrade and Versions** screen.

2. Guidewire disables (makes inactive) all updates to product model patterns during a rolling upgrade in order to preserve data integrity. After you complete the rolling upgrade, you must run Product Model Pattern Activation batch processing to make active all product model patterns that you added in the upgrade process.
3. PolicyCenter upgrade logic automatically triggers the activation of product model patterns added during a rolling upgrade if any node in the cluster restarts after the rolling upgrade completes. However, Guidewire does not recommend that you uses this mechanism to activate product patterns. Run Product Model Pattern Activation batch processing instead.
4. Guidewire specifically does not support the deletion of product model patterns of any type of upgrade.
5. Guidewire specifically does not support adding lookups with a status of Unavailable during a rolling upgrade.

See also

- "Product Model Pattern Activation batch process" on page 148.

Coverage, condition and exclusion patterns

PolicyCenter stores coverage, condition and exclusion patterns in XML files, in Studio directories of the following type under the `resources` directory:

```
productmodel/policylinepatterns/LOB/coveragepatterns/*.xml
```

PolicyCenter stores the coverage term and coverage term option patterns associated with each clause in a specific XML file for its parent clause. In the file path:

- *`LOB`* refers to a specific line of business, for example, `PersonalAutoLine`.
- `*` represents the coverage or exclusion pattern, for example, `PACollisionCov.xml` or `ExcludeCustomEquipment.xml`.

Guidewire supports only the addition of patterns for these types (coverages, conditions, exclusions, and their associated coverage term and coverage term options) during a rolling upgrade. Do not attempt to modify the attributes of any existing pattern of these types during a rolling upgrade. It is possible, however, to add new children to existing patterns. For example, it is possible to add a new coverage term to an existing coverage, or, a new coverage term option to an existing coverage term.

Offering patterns

PolicyCenter stores offering pattern data in XML files, in Studio directories of the following type under the `resources` directory:

```
productmodel/products/product/offerings/*.xml
```

In the file path:

- *`product`* refers to a specific product, for example, `BusinessAuto`.
- `*` represents the offering pattern, for example, `BAStandardOffering.xml`.

Guidewire supports only the addition of Offering patterns during a rolling upgrade. It is possible to add any type of offering selection to an Offering pattern during the upgrade.

Question set and question patterns

PolicyCenter stores question and question set pattern data in XML files, in the following Studio directory under the `resources` directory:

```
productmodel/questionsets/*.xml
```

In the file path:

- `*` represents the question set pattern, for example, `BABusinessAutoPreQualA.xml`

Guidewire supports only the addition of question set patterns or questions during a rolling upgrade. Do not attempt to modify or delete any existing question or question set pattern during a rolling upgrade. Guidewire specifically does not support the addition, modification or deletion of question answers in a product model rolling upgrade.

Modifier patterns

PolicyCenter stores modifier patterns in XML files, in Studio directories of the following types under the `resources` directory:

```
productmodel/products/product/product.xml
productmodel/policylinepatterns/LOB/LOB.xml
```

In the file path:

- *`product`* refers to a specific product, for example, `BusinessAuto` in the `products` directory. For example, you find the modifier patterns associated with the Business Auto product in the file of the same name, `BusinessAuto.xml`, in that directory.
- *`LOB`* refers to a specific line of business, for example, `BusinessAutoLine` in the `policylinepatterns` directory.

Guidewire supports only the addition of modifier patterns during a rolling upgrade.

Lookup, grandfathering, and modifier minmax data

IMPORTANT Guidewire specifically does not support adding lookups with a status of Unavailable during a rolling upgrade.

PolicyCenter stores grandfathering and jurisdiction-specific modifier MinMax data in XML files, in Studio directories of the following types under the `resources` directory:

```
productmodel/policylinepatterns/LOB/coveragepatterns/*-grandfathering.xml
productmodel/policylinepatterns/LOB/jurisdictions/jurisdiction/coveragepatterns/*-grandfathering.xml
productmodel/policylinepatterns/product/jurisdictions/jurisdiction/*-modifierminmax
productmodel/policylinepatterns/LOB/jurisdictions/jurisdiction/*-modifierminmax
```

In the file path:

- *`product`* refers to a specific product, for example, `BusinessOwners`.
- *`LOB`* refers to a specific line of business, for example, `BOPLine`.
- *`jurisdiction`* refers to the two character state code for the specific jurisdiction to which the file applies, for example, AK for Alaska.
- `*` represents the pattern for which the grandfathering and modifier MinMax file applies, for example, *`BAComprehensive`*`-grandfathering.xml` or *`PersonalAutoLine`*`-modifierminmax.xml`.

Lookup files are all of the form `*-lookups.xml`, with `*` as the name of the pattern associated with the lookup, for example, *`BAComprehensive`*`-lookups.xml`. PolicyCenter stores lookup files in multiple directories, in either of the following:

- In the same directory as the associated pattern
- In the appropriate jurisdiction-specific directory if the lookup is specific to a particular jurisdiction

For example, PolicyCenter stores non-jurisdiction-specific lookup rows for the `BAComprehensive` coverage pattern (or, any of its child coverage terms and coverage term options) in file *`BAComprehensive`*`-lookups.xml`. PolicyCenter locates this file in the following Studio directory:

```
proudctmodel/policylinepatterns/LOB/BusinessAutoLine/coveragepatterns
```

Correspondingly, PolicyCenter stores Alaska-specific lookups for the `BAComprehensive` coverage in the following directory in Studio:

```
productmodel/policylinepatterns/BusinessAutoLine/jurisdictions/AK/coveragepatterns
```

Guidewire supports modifications to the files in these directories in the following ways only:

- By adding new lookup, grandfathering, and modifier minmax files to the directory
- By adding new lookup and grandfathering elements to existing files

  Note: It is not necessary to use product model rolling upgrade to add new lookups or grandfathering availability data. It is possible also to update the PolicyCenter server with this information, as well as update availability scripts, using the Server Tools **Reload Availability Data** functionality. However, do not attempt to use this functionality during the rolling upgrade itself.

See also

- *Product Model Guide*

Verification of upgrade compatibility

Guidewire provides a command prompt build tool to use in determining if the source and target application configurations are sufficiently compatible to support the upgrade process. To use, run the following command from a command prompt on any cluster member.

```
system_tools -verifyconfig filepath
```

This tool compares the local server configuration with the configuration of the remote cluster. The *`filepath`* parameter defines the path to the local WAR/EAR file that contains the PolicyCenter application, `pc.war`, for example.

The tool provides an on-screen report that contains information about the feasibility of the upgrade changes for the server members in the cluster.

Source and target configurations are incompatible

If the source and target configurations are incompatible, Guidewire requires a full database upgrade. Guidewire does not permit a rolling upgrade using the target configuration. If a rolling upgrade of the cluster is not possible, the command lists the incompatible or missing files.

Source and target configurations are identical

If the source and target configurations are identical, no upgrade is necessary.

Source and target configurations are compatible

If the source and target configurations are compatible, Guidewire permits a rolling deployment of these changes.

Rolling upgrade in progress

If a rolling upgrade is in progress, there are several possible configurations active in the cluster. Running the -`verifyconfig` command option on a cluster member provides the following information about that cluster member:

- The cluster member is running the source configuration.
- The cluster member is running the target configuration.
- The cluster member is actively upgrading itself.

See also

- "system_tools command" on page 429

Performing a rolling upgrade

Performing a rolling upgrade on your production PolicyCenter cluster requires extensive testing to minimize the risk of deploying the upgrade changes in a production environment.

The following list describes the steps involved in performing a rolling upgrade.

Step	Task	For more information
Prepare	Verify that your environment is set up correctly for a rolling upgrade.	"Prepare for a rolling upgrade" on page 205
Test	Test the PolicyCenter upgrade changes in a test environment before you implement the changes in a production environment.	"Perform a rolling upgrade in a test environment" on page 206
Upgrade	Upgrade the servers in your production environment, one by one.	"Perform a rolling upgrade in a production environment" on page 207

Rolling upgrade terminology

Guidewire uses the following term definitions in the discussion on rolling upgrade.

Instance	An individual PolicyCenter server running in a VM (Virtual Machine) or JVM (Java Virtual Machine) or stand-alone PolicyCenter server.
Test instance	A PolicyCenter instance with the same data model and EAR/WAR build file as that used on a production instance. The test cluster does not need to have the same number of test instances as the production cluster. However, there needs to be at least two instances in the test cluster to be able to test the rolling upgrade process. Guidewire recommends that you place the servers in the test instance in production mode so as to replicate your production environment as much as possible.
Production instance	A member of the production cluster accessed and used by external PolicyCenter users.

Prepare for a rolling upgrade

Before you begin

Before you begin the rolling upgrade process, review "Performing a rolling upgrade" on page 204.

About this task

The following list describes the necessary actions that you need to undertake before performing the rolling upgrade.

Task	How?	More information
Verify that the database schema matches the PolicyCenter data model as defined in the data model metadata files.	Use the database schema verification tool available from the Server Tools **Database Table Info** screen.	"The Database Table Info screen" on page 369
Verify that the source and target changes are compatible.	Use the `system_tools -verifyconfig` utility to verify compatibility between the source and target PolicyCenter builds.	"Verification of upgrade compatibility" on page 204
Back up your current (source) application build.	Create a robust database backup and restore strategy in case there are problems upgrading individual cluster members.	Consult with your database administrator
Create a custom version label for your target deployment.	This step is optional. However, a version label provides a way to identify this new deployment for future reference.	"Understanding Guidewire software versioning" on page 400

Next steps

After you complete these steps, continue to "Perform a rolling upgrade in a test environment" on page 206.

Perform a rolling upgrade in a test environment

Before you begin

Before you begin testing the new PolicyCenter build with the upgrade changes, review "Prepare for a rolling upgrade" on page 205.

Procedure

1. In file `database-config.xml`, verify that the `<database>` element `autoupgrade` attribute is set to `manual` (or non-existent).

   If the attribute is missing, the default value for this attribute is `manual`. The value cannot be `full`.
2. Create a new EAR/WAR PolicyCenter build that includes all the proposed changes.

   Guidewire recommends that the build name include some identification such as a date or a version number.
3. Place the WAR/EAR build in a local directory on a test instance.

   Guidewire recommends that you place all servers in the test instance in production mode so as to replicate your production environment as much as possible.
4. Run the following command option from the command prompt.

   ```
   system_tools -verifyconfig filepath
   ```

   The verification tool compares the local server configuration with the configuration of the remote cluster. The *`filepath`* parameter defines the path to the local WAR/EAR file that contains the PolicyCenter build, `pc.war`, for example. If the tool indicates that the proposed changes are compatible with the source build, continue to the next step.
5. Back up the test database.
6. On any server instance in the cluster, navigate to the Server Tools **Upgrade and Versions** screen and click **Start Rolling Upgrade**.
7. Bring down one of the servers in the test environment and deploy the new WAREAR file to this server instance.
8. Bring the test instance back up.
9. Perform user acceptance testing on the test cluster, on servers that run either the source or target builds.

   If testing indicates that there are no major issues in running the two builds simultaneously, continue to the next step.
10. Deploy the new WAR/EAR file to all test instances.
11. Navigate to the Server Tools **Upgrade and Versions** screen of any server instance in the test environment.
12. Click **Rolling Upgrade Complete**.

    This action clears the upgrade flag indicating that a rolling upgrade is in progress. The rolling upgrade is now complete.
13. Run Product Model Pattern Activation batch processing to activate the newly added product model patterns.

    Running this batch process makes active all product model patterns that you added in the rolling upgrade. See "Product Model Pattern Activation batch process" on page 148 for more information.

    IMPORTANT Restarting any node in the cluster after you complete the rolling upgrade (after you click **Rolling Upgrade Complete**) automatically triggers the activation of the added product model patterns. Guidewire does not recommend that you activate the product model patterns in this manner.
14. Perform acceptance testing on all the test instances to verify that PolicyCenter works as intended.

Next steps

If testing indicates that there are no issues with the new PolicyCenter build running on all test instances, continue to "Perform a rolling upgrade in a production environment" on page 207.

Perform a rolling upgrade in a production environment

Before you begin

Before performing a rolling upgrade in a production environment, ensure that you have completed all steps in "Perform a rolling upgrade in a test environment" on page 206.

Procedure

1. In file `database-config.xml`, verify that the `<database>` element `autoupgrade` attribute is set to `manual` (or non-existent).

   If the attribute is missing, the default value for this attribute is `manual`. The value cannot be `full`.
2. On any instance in the PolicyCenter production cluster, navigate to the Server Tools **Upgrade and Versions** screen.
   a. Click **Start Rolling Upgrade**.
   b. Verify the checklist of upgrade prerequisites.
   c. Click **Start Rolling Upgrade**.

      This action sets a flag in the PolicyCenter database that indicates a rolling upgrade is in progress.
3. Navigate to the Server Tools **Cluster Members** screen on any server instance.
   a. For the instance that you want to shutdown, click **Start Planned Shutdown** in the **Actions** column.
   b. Set the appropriate shutdown parameters in the **Schedule Planned Shutdown** screen.
   c. Click **Schedule Shutdown**.

      This action schedules a shutdown of the specified instance. All users logged into this PolicyCenter instance see an on-screen message indicating that a planned shutdown is in progress. After the scheduled period of time elapses, there are no more user connections to this production instance.
   d. Manually shut down the application server that you scheduled for shutdown..
4. Deploy the target PolicyCenter build to the production instance that you shut down.
5. Bring the instance with the target build back up.
6. Perform acceptance testing on the production instances that use the target build to determine if there are any major issues with running the source and target builds in the same cluster.

   If testing indicates that there are no major issues with running both build types simultaneously in the production cluster, continue to the next step. If there are issues, repeat the previous steps until you have upgraded all the production instances with the target build.
7. Perform another round of acceptance testing.

   If testing indicates that there are no major issues with the target build running on all production instances, continue to the next step.
8. Navigate to the Server Tools **Upgrade and Versions** screen of any instance in the PolicyCenter production cluster.
9. Click **Rolling Upgrade Complete**.

   This action clears the upgrade flag indicating that a rolling upgrade is in progress. The rolling upgrade process is now complete.
10. Run Product Model Pattern Activation batch processing to activate the newly added product model patterns.

    Running this batch process makes active all product model patterns that you added in the rolling upgrade. See "Product Model Pattern Activation batch process" on page 148 for more information.

    IMPORTANT Restarting any node in the cluster after you complete the rolling upgrade (after you click **Rolling Upgrade Complete**) automatically triggers the activation of the added product model patterns. Guidewire does not recommend that you activate the product model patterns in this manner.
11. Perform another round of acceptance testing to ensure that there are no issues with the upgrade changes.

Unexpected upgrades

Any time that you deploy a new WAR/EAR file to a PolicyCenter server and restart the server, PolicyCenter assumes that an upgrade is in progress. To prevent the unexpected upgrade of a server, Guidewire requires that you set an upgrade flag in PolicyCenter before starting either a full or rolling upgrade.

Guidewire requires the use of this flag to mitigate the risk of accidentally triggering an unexpected upgrade. However, as a consequence, it is possible to encounter situations in which the PolicyCenter server does not start. In that case, you must undertake a recovery sequence to return the server to a state in which it can start.

Full application database upgrade

For a full database upgrade, Guidewire requires that you first set a database flag to indicate that a full upgrade is in progress. This action signals your intention to perform a full upgrade. After you complete the upgrade of all servers in the cluster, PolicyCenter deletes the database flag automatically. You must set the upgrade flag again before starting a new full upgrade.

It is possible to set the full upgrade in progress flag in the following ways.

PolicyCenter	To set the upgrade flag in PolicyCenter, click **Start Full Upgrade** in the Server Tools **Upgrade and Versions** screen.
System tools	To set the upgrade flag through system tools, use the following command option: `system_tools -startfullupgrade` At least one cluster member must be running in order for you to use this option.
Web services	To set the upgrade flag using web services, call the `SystemToolsAPI` web service method `startFullUpgrade`. At least one cluster member must be running in order for you to use this option.
Java system property	To set the upgrade flag through a Java system property, use the following system property to set the expected date of the upgrade while starting one of the affected servers: `gwb runServer -Dgw.pc.full.upgrade.intended.date=`*`date`* The *`date`* property is the current date in `yyyyMMdd` format.
File `database-config.xml`	To set the upgrade flag on server start, set the `autoupgrade` attribute on the `<database>` element in file `database-config.xml`.

If you encounter a situation in which all cluster members refuse to start because the upgrade flag was not set, you cannot set the upgrade flag through the server. Instead, you must use one of the following methods to start the full upgrade:

- Set the upgrade flag using the Java system property.
- Set the `autoupgrade` attribute in file `database-config.xml`.

Both of these methods provide the necessary permission to perform a full database upgrade of the cluster members.

Rolling upgrade

For a rolling upgrade, Guidewire first requires that you click **Start Rolling Upgrade** in the **Upgrade and Versions** screen (on any cluster member). This action signals your intention to perform a rolling upgrade and sets a rolling upgrade in progress database flag. If you do not set the upgrade flag, PolicyCenter refuses to start a rolling upgrade.

After you complete the upgrade of all servers in the cluster, you must click **Rolling Upgrade Complete** on the **Upgrade and Versions** screen, which removes the upgrade flag. After you do so, it is not possible start a cluster member running the source (old) configuration.

Guidewire permits a rolling upgrade of the individual members of a PolicyCenter cluster under certain conditions only. In effect, the source (old) configuration and target (new) configuration must be compatible in very specific ways.

Thus, during a rolling upgrade, if you mistakenly deploy an incompatible WAR/EAR file to a PolicyCenter server, you can encounter a situation in which the server does not start. This is true whether you have set the rolling upgrade in progress flag. In this case, remove the incompatible WAR/EAR file and deploy a compatible WAR/EAR file before attempting to restart the server.

See also

- “Supported rolling update changes” on page 199
- “Verification of upgrade compatibility” on page 204

Upgrade of a production stand-alone PolicyCenter server

A stand-alone server has configuration parameter `ClusteringEnabled` set to `false`. The use of a rolling upgrade is not possible on a stand-alone server. However, while some upgrade changes require a full database upgrade, other upgrade changes do not require a full upgrade.

A full upgrade on a production PolicyCenter stand-alone server causes the following behavior.

Upgrade type	Example	Behavior
Full	Changing the value of a semi-permanent configuration parameter. PolicyCenter writes these kinds of changes to the application database.	If the flag for a full database upgrade is not set, the application server does not start. If flag for a full database upgrade is set, the upgrade completes and PolicyCenter updates the **Upgrade and Versions** screen.
Supported changes	Changing the value of a general configuration parameter, one that is neither permanent nor semi-permanent. PolicyCenter reads these kinds of changes from configuration files.	It does not matter whether you set the flag for a full database upgrade. In any case, the application server starts without any errors or warnings and PolicyCenter updates the **Upgrade and Versions** screen.

Backing out a rolling upgrade

It is possible to back out configuration changes on a given server application instance, depending on the state of the configuration deployment on that instance. Guidewire does not recommend that you back out an ongoing deployment on a server instance except in extraordinary circumstances. If you do so, you need to test that the server configuration returns to a safe state.

To initiate a back out of a rolling upgrade, click **Initiate Backout** on the Server Tools **Upgrade and Versions** screen. See “Performing server upgrades” on page 402 for more information.

Before you start the rolling upgrade

In all cases, Guidewire recommends that you run the database schema verifier before every rolling upgrade, especially if the upgrade contains data model schema changes. To verify the data model schema against the database, use the following `system_tools` command option:

```
system_tools -verifydbschema
```

Fix any issues identified during the schema verification process before you begin the rolling upgrade

Preventing table writes during a rolling upgrade

To support backing out a rolling upgrade, Guidewire prevents a user from writing to any table column newly added during the upgrade process until the rolling upgrade is fully complete:

- During the rolling upgrade, PolicyCenter tracks which table columns the upgrade process adds and stores the list of new columns in the database.
- For all cluster servers, PolicyCenter reads the list on server startup and throws an exception for that server if a user tries to write to a newly added column before the rolling upgrade is complete.
- It is only after the rolling upgrade completes (a user clicks **Rolling Upgrade Complete** in the Server Tools **Upgrade and Versions** screen) that PolicyCenter permits write operations to any table column added during the rolling upgrade.

Enforcing this restriction makes rolling back data model schema changes easier as the rollback process does not have to contend with data inconsistencies between the different cluster nodes.

Backing out schema changes

The back out process does not undo any schema changes made during the rolling upgrade process other than changes to typelists.

Rolling upgrade and business rules

During a rolling upgrade, Guidewire disables the ability to deploy a rule in a production environment. The intent is to prevent the deployment of a rule that contains data that is incompatible with the current environment, thereby causing the rule execution to fail.

During a rolling upgrade, PolicyCenter:

- Disables the **Deploy All** and **Deploy Selected** options on the rule list screen.
- Disables the **Deploy** button on the rule detail screen.
- Displays an alert message on the rule list and rule details screen that indicates rule deployment is not possible due to an ongoing upgrade.

Thus, during a rolling upgrade, Guidewire hides the **Deploy** functionality.

part 4

Security administration

chapter 13

Managing secure communications

Guidewire products use a 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 a PolicyCenter server to a level strong enough that it cannot be easily compromised.

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 perform independent research and testing to develop a secure solution for your company network and installed applications. Guidewire strongly recommends that you deploy PolicyCenter over TLS (Transport Layer Security) for at least the login and change password pages. Ideally, deploy PolicyCenter entirely under TLS to protect all sensitive transmitted data.

PolicyCenter And the Transport Layer Security protocol

Guidewire applications use the TLS (Transport Layer Security) protocol while making WS-I and RPC web service connections to HTTPS endpoints. TLS is the only communication protocol that Guidewire supports for this purpose. There are multiple versions of the TLS protocol. All current releases of JDK 7 support TLS 1.0, TLS 1.1, and TLS 1.2.

The TLS default version is the underlying JDK default for the installed release of JDK 7. This is TLS 1.0 for the public, free (non-paid support), releases. However, use of the TLS 1.0 default can cause the connection to a server to fail if the server requires either TLS 1.1 or TLS 1.2.

Setting TLS version overrides

Guidewire provides several Java property overrides to set the default TLS version to use on outgoing secured connections. You can use these property overrides with either the paid support or the free versions of JDK 7. Use these property overrides to provide a comma-separated list of TLS protocol versions. PolicyCenter uses the first item on the list as the preferred protocol. If that protocol is not available, PolicyCenter tries the subsequent protocols on the list until the connection either succeeds or fails completely.

The following table lists the available property overrides.

Web service type	Property	Syntax
WS-I	`gw.webservices.tls.protocols`	`-D.gw.webservices.tls.protocols="a, b"`
RPC	`gw.tls.protocols`	`-Dgw.tls.protocols="a, b"`

In the table, `a` and `b` refer to TLS versions, for example:

```
<java> ... -D.gw.webservices.tls.protocols="TLSv1.2, TLSv1.1"
```

Notice the following for this example:
- The property definition indicates that TLS1.2 is the preferred protocol. However, if TLS1.2 is not available, PolicyCenter attempts to use TLS 1.1 instead.
- The property definition affects only client WS-I web service calls.

PolicyCenter and secure communications

A strong password policy is the first and best line of defense. Guidewire also recommends the following:
- Encrypt the communication between the Internet and the PolicyCenter server or cluster.
- Configure a separate server to act as an intermediary layer between the Internet and any PolicyCenter server or servers. Typically, you locate this intermediary server in a DMZ that you establish through your network architecture.

If you off-load encryption to a server, understand that non-native encryption processing is not as efficient. Native applications generally use optimized encryption modules.

You can use a web server or proxy both to encrypt communications and to provide a layer between the Internet and a PolicyCenter server. Computer network terminology generally calls a server working as an intermediary in this manner a reverse proxy. There are multiple methods you can use to achieve an encrypted proxy solution.

See also

- “The PolicyCenter connection address” on page 214

The PolicyCenter connection address

In setting up a secure connection with Guidewire PolicyCenter, ensure that you use a connection address similar to the following:

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

You need to set the server name and port number to that on which the proxy server is running. Notice the use of the `https` protocol instead of `http`, indicating that the server is connecting through a secure version of HTTP.

Set this address for every client that connects to the PolicyCenter application server, including web browsers, PolicyCenter plugins, and applications that use PolicyCenter APIs.

In configuring this URL, also check PolicyCenter file `config.xml` for any URL specifications that you need to specify.

Restricting access to a PolicyCenter screen by environment

Guidewire uses PCF (Page Configuration Format) files to render PolicyCenter screens. Properties associated with each PCF screen determine whether it is possible to visit, or edit, the screen.

It is possible to restrict access to a screen based on server environment. For example, an administrator can usually access an **Activity Patterns** screen. This screen provides a list of all of the activity patterns available in PolicyCenter. Clicking one of the listed activity patterns opens that pattern's detail screen.

Suppose, for some reason, that you want to restrict access to the **Activity Patterns Detail** screen on a production system. Opening up `ActivityPatternDetails.pcf` in Studio shows the following properties:

- **canEdit**
- **canVisit**

To permit access to the **Activity Patterns Detail** screen on a production system only, enter the following value for the **canVisit** property:

```
gw.api.system.server.ServerUtil.getEnv() == "PROD"
```

The **canVisit** property must evaluate to `true` for the **Activity Patterns Detail** screen to be accessible to a PolicyCenter administrative user. If the server is in development or test mode, the expression evaluates to false and PolicyCenter restricts access to the screen.

chapter 14

Securing access to PolicyCenter objects

Guidewire designs its security infrastructure so that you can add custom permissions, automatically enforce permissions, and easily map between users, permissions, and actions. This topic explains how to use the PolicyCenter permission infrastructure to control access to key PolicyCenter objects.

Understanding the object access infrastructure

You can assign each user one or more roles that contain permissions. These permissions control what the user can do in Guidewire PolicyCenter. For example, a user in PolicyCenter with the correct role can create a note on a Policy. The limitation of roles is that they do not distinguish among objects of the same type. In the previous example, “note” means all notes and all note types. However, suppose that you want to restrict access to notes that contain sensitive information. In this case, PolicyCenter provides access control features that you use to restrict access to specific types of notes.

By implementing access control, you can subcategorize an object type and then restrict object access by these subcategories. In the base configuration, you can apply access control to the following business objects:

- `Account`
- `Activity`
- `Document`
- `Job` – and all its subtypes
- `Note`
- `PolicyPeriod`
- `User`

It is possible to apply access control (permissions) to any PolicyCenter business object.

Understanding the PolicyCenter permission types

PolicyCenter contains the following types of permissions:

- *System permission*, which apply to specific user interface elements or data model entities.
- *Application permission*, which represent a set of one or more system permissions.

You can view a list of both system and application permission keys in the Guidewire *Security Dictionary*.

See also

- *Configuration Guide*

System permission keys

Guidewire groups system permissions into the following categories:

- Screen-level permissions that apply to user interface elements
- Domain-level permissions that apply to data model entities

You can view a list of system permissions in the `SystemPermissionType` typelist. You can also extend this typelist and add custom permissions.

Screen-level permissions

Screen-level permissions apply to user interface elements, for example, the permission to view the administrative **Server Tools** screens. PolicyCenter defines many user interface permissions internally.

In general, screen-level permissions start with the word “view” followed by a reference to the user interface object they protect. You can add custom screen-level permissions to Guidewire PolicyCenter by extending the `SystemPermissionType` typelist.

PCF files define the point at which PolicyCenter calls user interface permissions. It is possible to change this point by customizing the PCF file that calls it.

Domain-level permissions

Domain permissions apply to data model entities, such as permission to view `Note` objects. For example, as a user attempts to access the summary for a sensitive note, PolicyCenter verifies that the user has the following permissions:

- Permission to view the **Policy** screen
- Permission to access that particular note type

Most top-level objects in the PolicyCenter data model have associated domain-level permissions. PolicyCenter defines all of an object’s domain-level permissions internally. It is not possible to add, remove, or edit domain permissions. Similarly, PolicyCenter defines the points at which it checks these permissions in internal code and in page configuration format (PCF) files. You cannot change the internal checks. You can, however, change the point at which the PCF files calls these checks.

Application permission keys

Application permission keys represents a set of one or more system permissions. PolicyCenter defines application permission keys internally as a method for improving system performance. For example, the `Activity own` permission key represents the system permission for owning an activity. The `Activity edit` permission key represents the system permission for the editing activities.

Guidewire defines all configurable application permissions in file `security-config.xml`. It is possible for you to modifying this file and add new application permissions.

Guidewire defines many other access application permissions internally. It is not possible to change these permissions.

See also

- “The security configuration file” on page 219
- *Configuration Guide*

Permission class generation

PolicyCenter generates its permission classes at server startup by parsing file `security-config.xml`. The PolicyCenter log shows this activity:

```
...INFO SecurityManager finished parsing ...\modules\configuration\config\security\security-config.xml
```

PolicyCenter does not generate permissions automatically for the subtypes of an entity. You must explicitly add the entity subtype to `security-config.xml` for PolicyCenter to generate permissions for that subtype.

Beyond roles and permissions to access control

You can group system permissions by adding them to roles and then assigning the role to a user. So, if a particular role has a view policy document permission, any user with that role can view a document attached to a policy. And, of course, the user must first have access to that policy.

In practice, however, you likely do not want all users to access all objects of the same type. For example, suppose that an object has an associated document that contains information on a famous celebrity. You most likely want to restrict access so that only certain people have access to the personal information contained in this document. You use the PolicyCenter access control feature to make distinctions among objects of the same type and then secure access to them.

While roles and permissions determine what actions a user can perform, access control determines the objects on which the user can act. After you enable access control, a user requires both the correct role and the proper access. To use access control, you apply a security attribute to an object and then determine which users have access to objects with that attribute.

Access control configuration files

You manage access control in PolicyCenter with the following files:

File	Role the file plays in access control
`security-config.xml`	Defines business object security handlers.
`DocumentSecurityType.ttx`	Defines access control types related to documents.
`MoneySecurityType.ttx`	Defines access control types related to money.
`NoteSecurityType.ttx`	Defines access control types related to notes.
`SystemPermissionType.ttx`	Contains customizable and custom system permissions.

You modify these files in Guidewire Studio.

- File `config.xml` is available under **configuration→config**.
- File `security-config.xml` is available are under **configuration→config →security**.
- The various typelist TTX files are available under **configuration→config→Metadata→Typelist**.

See also

- “Understanding the object access infrastructure” on page 217

The security configuration file

PolicyCenter provides the means to specify user access to business objects through application configuration file `security-config.xml`. This XML-formatted file contains a number of XML elements that you use to configure user access to specific business objects.

You access `security-config.xml` in Guidewire Studio, in the following location:

configuration→config →security

Object access and security

The following topics discuss the standard XML elements in `security-config.xml` that relate to PolicyCenter object access and security:

- "Static handler elements" on page 220
- "Wrap handler elements" on page 221

Note and document objects

The following topics discuss the XML elements in `security-config.xml` that relate to PolicyCenter note and document objects

- "Note permission overview" on page 225
- "Document permission overview" on page 226

Account, job and policy period objects

The following topics discuss the XML elements in `security-config.xml` that relate to PolicyCenter account, job, and policy period objects:

- "Account producer code handler elements" on page 229
- "Job producer code handler elements" on page 231
- "Policyperiod producer code handler elements" on page 232

See also

- "Understanding the object access infrastructure" on page 217
- "Access control configuration files" on page 219

Static handler elements

You use the `<StaticHandler>` element in `security-config.xml` to define security permissions on an entity. This type of security permission is static and requires no object.

There is no limit to the number of `<StaticHandler>` elements that can exist in `security-config.xml`. Each `<StaticHandler>` element can contain zero to many `<SystemPermType>` elements.

This element has the following syntax.

```
<StaticHandler entity="entity" permKey="perm" desc="..." noPermissionDisplayKey="key">
  <SystemPermType code="code"/>
  ...
</StaticHandler>
```

You access this permission in code as `perm.`*`entity`*`.`*`perm`*. This syntax has the following meaning:

- *`entity`* – The business object or entity on which the permission acts.
- *`perm`* – The permission given for this entity.

The attributes on the various elements have the following meanings.

Element	Attribute	Required	Description
`StaticHandler`	`entity`	Yes	The entity type on which this security handler acts.
	`permKey`	Yes	The application permission to grant.
	`desc`	No	A human-readable description of the permission.
	`noPermissionDisplayKey`	No	A display key that provides the text to show if the user does not have a required permission.
`SystemPermType`	`code`	Yes	A code value defined in the `SystemPermissionType` typelist.

The following example shows a typical `<StaticHandler>` element.

```
<StaticHandler entity="User" permKey="ViewProfiler" noPermissionDisplayKey="No access to ViewProfiler.">
  <SystemPermType code="internaltools"/>
  <SystemPermType code="toolsprofilerview"/>
</StaticHandler>
```

Notice that:

- The security permissions work on a `User` entity.
- The application permission key is `ViewProfiler`.
- The handler lists a set of specific system permission types to which the handler grants the user access, if any of the conditions are met.

To have the `ViewProfiler` application permission, the user must have an assigned role that contains one or more of the listed system permissions.

Static handlers specify OR conditions

Static security handlers define Boolean `OR` conditions. This means for the user to have a certain application permission, the user must have an assigned role that contains at least one of the following:

- System permission A
- Or, system permission B
- Or, system permission C
- Or, …

Suppose that you have the following code that references the `ViewProfiler` static handler shown previously.

```
if (perm.User.ViewProfiler) ...
```

The sample code condition evaluates to `true` if the current user has an assigned role with either the `internaltools` permission or the `toolsprofilerview` permission.

See also

- “The security configuration file” on page 219
- “Wrap handler elements” on page 221

Wrap handler elements

The `<WrapHandler>` element in `security-config.xml` defines complex security permissions on an entity. The wrap handler “wraps around” the permission conditions of the associated handler. The associated handler type must not be object-based, meaning that it must be one of the following:

- `<StaticHandler>`
- `<WrapHandler>`

It is not possible to create a security handler that takes an object using a `<WrapHandler>` element. A wrap security handler always create a new static handler.

There is no limit to the number of `<WrapHandler>` elements that can exist in `security-config.xml`. Each `<WrapHandler>` element can contain zero to many `<SystemPermType>` elements.

A `<WrapHandler>` element must come after the `<Handler>` element that defines the permission referenced by `wrapPermKey`. The associated handler can be another `<WrapHandler>`. It is possible to cascade `<WrapHandler>` elements.

This element has the following syntax.

```
<WrapHandler entity="entity" permKey="perm" wrapPermKey="wrapPerm" desc="..." noPermissionDisplayKey="key">
  <SystemPermType code="code"/>
  ...
</WrapHandler>
```

You access this permission in code as `perm.`*`entity`*`.`*`perm`*. This syntax has the following meaning:

- *`entity`* – The business object or entity on which the permission acts.
- *`perm`* – The permission given for this entity.

The attributes on the various elements have the following meanings.

Element	Attribute	Required	Description
`WrapHandler`	`entity`	Yes	The entity type on which this security handler acts.
	`permKey`	Yes	The application permission to grant.
	`wrapPermKey`	Yes	The associated permission being wrapped. You must declare the permission referenced by the `wrapPermKey` earlier in `security-config.xml` than this `<WrapHandler>` element.
	`desc`	No	A human-readable description of the permission.
	`noPermissionDisplayKey`	No	A display key that provides the text to show if the user does not have a required permission.
`SystemPermType`	`code`	Yes	A code value defined in the `SystemPermissionType` typelist.

The following example illustrates a `<StaticHandler>` element with two cascading `<WrapHandler>` elements following it.

```
// Static Handler - ViewProfiler permission
<StaticHandler entity="User" permKey="ViewProfiler" noPermissionDisplayKey="No access to ViewProfiler.">
  <SystemPermType code="internaltools"/>
  <SystemPermType code="toolsProfilerview"/>
</StaticHandler>

 //First Wrap Handler - EditProfiler permission
<WrapHandler entity="User" permKey="EditProfiler" wrapPermKey="ViewProfiler" noPermissionDisplayKey="No access to
EditProfiler.">
  <SystemPermType code="internaltools"/>
  <SystemPermType code="toolsProfileredit"/>
</WrapHandler>

 //Second Wrap Handler - EditWebserviceProfiler permission
<WrapHandler entity="User" permKey="EditWebserviceProfiler" wrapPermKey="EditProfiler" noPermissionDisplayKey="No
access to EditWebServiceProfiler.">
  <SystemPermType code="toolsProfilerwebserviceedit"/>
</WrapHandler>
```

This sequence of handlers does the following:

1. The first wrap handler verifies that the user meets the security criteria defined in the handler specified by its `wrapPermKey` attribute (`ViewProfiler`). If the user has an assigned role that contains any of the system permissions specified by the `ViewProfiler` handler, the handler permits the user to have the `EditProfiler` application permission. If the user does not have such a role, she receives an error message.
2. The second wrap handler checks to see that the user meets the security criteria defined in the handler specified by its `wrapPermKey` attribute (`EditProfiler`). If the user has an assigned role that contains the `toolsProfilerwebserviceedit` permission, the handler permits the user to have the `EditWebserviceProfiler` application permission. If the user does not have such a role, she receives an error message.

Wrap handlers specify AND conditions

Wrap security handlers define Boolean `AND` conditions. Using the example shown previously, the sequence of security handlers evaluates the following set of conditions:

```
(perm.System.internaltools OR perm.System.toolsProfilerview)
AND (perm.System.internaltools OR perm.System.toolsProfilerEdit)
AND (permission.System.toolsProfilerwebserivceedit)
```

For this compound condition to evaluate to `true`, all of the following conditions must be true:

- The user must have a role that contains either the `interntools` or `toolsProfilerview` system permission as specified in `ViewProfiler` static handler.
- The user must have a role that contains either the `interntools` or `toolsProfileredit` system permission as specified in the `EditProfiler` wrap handler.
- The user must have a role that contains the `toolsProfilerwebservicesedit` system permission as specified in the `EditWebServiceProfiler` wrap handler.

Only if the user meets all sets of security criteria does the security handler permit the user to have the specified application permission (`EditwebserviceProfiler`).

See also

- "The security configuration file" on page 219
- "Static handler elements" on page 220

chapter 15

Securing access to notes and documents

This topic explains how to use the PolicyCenter permission infrastructure to control access to document and note objects.

See also

- “Understanding the object access infrastructure” on page 217
- “The security configuration file” on page 219
- “Static handler elements” on page 220
- “Wrap handler elements” on page 221

Note permission overview

In addition to using the standard system permissions for notes, you can control access to notes by configuring note access permissions in `security-config.xml`. To use this feature, a note must have a note security type set. To see a note of a particular type, a user must have both permission to view notes generally and the permission to access to the note security type.

You set the security type for a note type by setting the note **Security Type** in PolicyCenter or through a Gosu class that you write. In the base configuration, PolicyCenter provides the following note security types in the `NoteSecurityType` typelist:

- Internal Only
- Sensitive
- Unrestricted

See also

- For information on the various security handler elements, see “The security configuration file” on page 219.
- For information on permissions, refer to the system permissions area of the *PolicyCenter Security Dictionary*.
- For information on typelists, refer to Guidewire Studio™ for PolicyCenter, or the typelists area of the *PolicyCenter Data Dictionary*.

Note permission elements

File `security-config.xml` must contain a `<NoteAccessProfile>` element for every note security type listed in the `NoteSecurityType` typelist. If you add a new note security type to the typelist, then you must add a corresponding `<NoteAccessProfile>` element to `security-config.xml`.

Thus, a `<NotePermission>` element in the `security-config.xml` file controls access to a note type.

The `<NotePermission>` element has the following syntax:

```
<NotePermissions>
  <NoteAccessProfile securitylevel="level">
    <NoteCreatePermission permission="perm"/>
    <NoteDeletePermission permission="perm"/>
    <NoteEditBodyPermission permission="perm"/>
    <NoteEditPermission permission="perm"/>
    <NoteViewPermission permission="perm"/>
   </NoteAccessProfile>
</NotePermissions>
```

The attributes on the various elements have the following meanings.

Element	Attribute	Required	Description
`NoteAccessProfile`	`securitylevel`	Yes	A document security type defined in the `NoteSecurityType` typelist.

`NoteCreatePermission`
`NoteDeletePermission`
`NoteEditBodyPermission`
`NoteEditPermission`
`NoteViewPermission`
	`permission`	Yes	A system permission defined in the `SystemPermissionType` typelist.

The following code sample illustrates the security access levels for the Sensitive security access type.

```
<NotePermissions>
  <NoteAccessProfile securitylevel="sensitive">
    <NoteViewPermission permission="viewsensnote"/>
    <NoteEditPermission permission="editsensnote"/>
    <NoteDeletePermission permission="delsensnote"/>
  </NoteAccessProfile>
 </NotePermissions>
```

Note: PolicyCenter grants access permissions based on the roles assigned to a user only. It is not possible to restrict Note access based on security zones or groups.

Document permission overview

In addition to using the standard system permissions for documents, you can control access to documents by configuring document access permissions in `security-config.xml`. To use this feature, a document must have a document security type set. To see a set of documents of a particular type, a user must have both permission to view documents generally and access to the document security type.

You set a document type by using the document's **Security Type** on the user interface or through a Gosu class that you write. In the base configuration, PolicyCenter provides the following document security types in the `DocumentSecuritytype` typelist:

- Internal Only
- Sensitive
- Unrestricted

See also

- For information on security handler elements, see “The security configuration file” on page 219.
- For information on permissions, refer to the system permissions area of the *PolicyCenter Security Dictionary*.
- For information on typelists, refer to Guidewire Studio™ for PolicyCenter or the typelists area of the *PolicyCenter Data Dictionary*.

Document permission elements

File `security-config.xml` must contain a `<DocumentAccessProfile>` element for every document security type listed in the `DocumentSecurityType` typelist. If you add a new document security type to the typelist, then you must add a corresponding `<DocumentAccessProfile>` element to `security-config.xml`.

Thus, a `<DocumentPermission>` element in the `security-config.xml` file controls access to a document type.

This element has the following syntax:

```
<DocumentPermissions>
  <DocumentAccessProfile securitylevel="level">
    <DocumentCreatePermission permission="perm"/>
    <DocumentDeletePermission permission="perm"/>
    <DocumentEditPermission permission="perm"/>
    <DocumentViewPermission permission="perm"/>
  </DocumentAccessProfile>
</DocumentPermissions>
```

The attributes on the various elements have the following meanings.

Element	Attribute	Required	Description
`DocumentAccessProfile`	`securitylevel`	Yes	A document security type defined in the `DocumentSecurityType` typelist.

`DocuumentCreatePermission`
`DocumentDeletePermission`
`DocumentEditPermission`
`DocumentViewPermission`
	`permission`	Yes	A system permission defined in the `SystemPermissionType` type-list.

The following code sample illustrates the security access levels for the Unrestricted and Internal Only security access types. Notice that unrestricted documents have no access controls set.

```
<DocumentPermissions>
  <DocumentAccessProfile securitylevel="unrestricted"/>
  <DocumentAccessProfile securitylevel="internalonly">
     <DocumentViewPermission permission="viewintdoc"/>
     <DocumentEditPermission permission="editintdoc"/>
     <DocumentDeletePermission permission="delintdoc"/>
     </DocumentAccessProfile>n="delsensdoc"/>
   </DocumentAccessProfile>
</DocumentPermissions>
```

Note: PolicyCenter grants these permissions based on the user’s roles alone. You cannot restrict document access based on security zones or groups.

chapter 16

Securing access to accounts, jobs, and policy periods

This topic explains how to use the PolicyCenter permission infrastructure to control access to PolicyCenter accounts, jobs and policy periods.

See also

- "Understanding the object access infrastructure" on page 217
- "The security configuration file" on page 219
- "Static handler elements" on page 220
- "Wrap handler elements" on page 221
- "Note permission overview" on page 225
- "Document permission overview" on page 226

Account producer code handler elements

`<AccountProducerCodeHandler>` elements in `security-config.xml` determine account access by controlling the account information and functionality to which a user has access. For example, an `<AccountProducerCodeHandler>` element can define who has permission to create an account for a particular produce code.

There is no limit to the number of `<AccountProducerCodeHandler>` elements that can exist in `security-config.xml`. Each `<AccountProducerCodeHandler>` element can contain zero to many `<SystemPermType>` and `<ProducerStatus>` elements.

This element has the following syntax.

```
<AccountProducerCodeHandler permKey="perm" desc="...">
  <SystemPermType code="code" />
  <ProducerStatus code="status" />
  ...
</AccountProducerCodeHandler>
```

The attributes on the various elements have the following meanings.

Element	Attribute	Required	Description
`AccountProducerCodeHandler`	`permkey`	Yes	The application permission to grant. The `permkey` attribute becomes a property on the `Account` entity. To access the entity property, use the following notation: `perm.Account.`*`permkey`*
	`desc`	No	A human-readable description of the permission.
	`noPermissionDisplayKey`	No	A display key that provides the text to show if the user does not have a required permission.
	`bypassProducerCodeCheckOnAccountsWithNoActivePolicies`	No	Skip producer code checks on accounts with no active policies.
`SystemPermType`	`code`	Yes	A code value defined in the `SystemPermissionType` typelist.
`ProducerStatus`	`code`	Yes	The status of the producer associated with the policy. This must be a value defined in the `ProducerStatus` typelist.

PolicyCenter typically limits account visibility to users who have one of the producer codes associated with the policies on the account. Producer status is an additional means of restricting access to account information. For example, you can use a producer status of `Suspended` to limit the actions that user can take on that account or the information that is visible on the account.

No restrictions

An `<AccountProducerCodeHandler>` element that does not contain a `<ProducerStatus>` element grants the specified permission without restriction to producers with a status is `Active`. The following example illustrates this concept.

```
<AccountProducerCodeHandler permKey="createForProducerCode"
      desc="Permission to create account for a particular producer code">
  <SystemPermType code="accountcreate"/>
</AccountProducerCodeHandler>
```

As there is no `<ProducerStatus>` element in this example, the permission handler grants the ability to create a new account to any producer with a status of `Active`.

Multiple producer status codes

The following example illustrates an `<AccountProducerCodeHandler>` element that contains multiple producer status codes.

```
<AccountProducerCodeHandler permKey="view" desc="Permission to view an account">
  <SystemPermType code="viewaccount"/>
  <ProducerStatus code="Limited"/>
  <ProducerStatus code="Suspended"/>
  <ProducerStatus code="Terminating"/>
</AccountProducerCodeHandler>
```

Notice that:

- The application permission key is `view`.
- The system permission type is `viewaccount`.
- The element explicitly grants permission to view an account to producers with a status of `Limited`, `Suspended`, or `Terminating`. PolicyCenter gives any producer with a status of `Active` this permission automatically. Thus, all producers can view accounts associated with their producer code except those producers with a status of `Terminated`. As always, producers can only view accounts associated with their producer code.

Skip producer code checks on account with no active policies

To skip producer code checks on accounts with no active policies, use the `bypassProducerCodeCheckOnAccountsWithNoActivePolicies` attribute in `security-config.xml`. When `true`, if all policies on the account are expired, any agent (regardless of agency) can view the account, and can start a new submission on that account. The account is treated as if there are no policies on it.

For example, to skip the producer code check on account while evaluating view account permission, add the following to `security-config.xml`:

```
<AccountProducerCodeHandler
    permKey="view"
    desc="Permission to view an account"
    bypassProducerCodeCheckOnAccountsWithNoActivePolicies="true">
```

To skip the check while evaluating edit account permission, add the following:

```
<AccountProducerCodeHandler
    permKey="edit"
    desc="Permission to edit an account"
    bypassProducerCodeCheckOnAccountsWithNoActivePolicies="true">
```

To skip producer code check on the **Account** search screen, set the `Secure` property on `AccountSearchCriteria` to `false`.

See also

- "The security configuration file" on page 219
- "Static handler elements" on page 220
- *Application Guide*

Job producer code handler elements

`<JobProducerCodeHandler>` elements in `security-config.xml` determine the access a producer has to the different job types in PolicyCenter. For example, an `<JobProducerCodeHandler>` element can define who has permission to create a submission job.

There is no limit to the number of `<JobProducerCodeHandler>` elements that can exist in `security-config.xml`. Each `<JobProducerCodeHandler>` element can contain zero to many `<SystemPermType>` and `<ProducerStatus>` elements.

This element has the following syntax.

```
<JobProducerCodeHandler jobType = "type" permKey="perm" desc="..." noPermissionDisplayKey="...">
  <SystemPermType code="code" />
  <ProducerStatus code="status" />
  ...
</JobProducerCodeHandler>
```

The attributes on the various elements have the following meanings.

Element	Attribute	Required	Description
`JobProducerCodeHandler`	`jobType`	Yes	The type of `Job` entity to which the permission applies, for example, a Submission or Renewal job.
	`permkey`	Yes	The application permission to grant. The `permkey` attribute becomes a property on the specified job type (`jobType`). To access the entity property, use the following notation: `perm.`*`jobType`*`.`*`permkey`*
	`desc`	No	A human-readable description of the permission.
	`noPermissionDisplayKey`	No	A display key that provides the text to show if the user does not have a required permission.
`SystemPermType`	`code`	Yes	A code value defined in the `SystemPermissionType` typelist.
`ProducerStatus`	`code`	Yes	The status of the producer associated with the policy. This must be a value defined in the `ProducerStatus` typelist.

The following example illustrates a `JobProducerCodeHandler` element.

```
<JobProducerCodeHandler jobType="Reinstatement" permKey="view"
       desc="Permission to view a reinstatement">
  <SystemPermType code="viewreinstate"/>
<ProducerStatus code="Limited"/>
```

Notice that:

- The value of `jobType` is `Reinstatement`.
- The application permission key is `view`.
- The system permission type is `viewreinstate`.
- The element explicitly grants permission to view the details of a policy reinstatement job to producers with a status of `Limited`. PolicyCenter gives any producer with a status of `Active` this permission automatically. Thus, only producers with a status of `Active` or `Limited` can view the details of reinstatement submission job.

See also

- "The security configuration file" on page 219
- "Static handler elements" on page 220
- *Application Guide*

Policyperiod producer code handler elements

`<PolicyPeriodProducerCodeHandler>` elements in `security-config.xml` determine access to policy information and restrict the actions available on a policy. There is no limit to the number of `<PolicyPeriodProducerCodeHandler>` elements that can exist in `security-config.xml`. Each `<PolicyPeriodProducerCodeHandler>` element can contain zero to many `<SystemPermType>` and `<ProducerStatus>` elements.

This element has the following syntax.

```
<PolicyPeriodProducerCodeHandler permKey="perm" desc="..."
      isAllowedForPCOfRecord="true|false" noPermissionDisplayKey="...">
```

```
  <SystemPermType code="code" />
  <ProducerStatus code="status" />
  ...
</PolicyPeriodProducerCodeHandler>
```

The attributes on the various elements have the following meanings.

Element	Attribute	Required	Description
`PolicyPeriodProducerCodeHandler`	`permkey`	Yes	The application permission to grant. The `permkey` attribute becomes a property on the `PolicyPeriod` entity. To access the entity property, use the following notation: `perm.`PolicyPeriod`.`*`permkey`*
	`desc`	No	A human-readable description of the permission.
	`isAllowedForPCOfRecord`	No	A Boolean value which, if `true`, grants this permission to a produce whose producer code matches that of the policy. The default is `false`. In the base configuration, Guidewire sets this attribute value to `true` for the `viewpolicyfile` permission only. Thus, only a producer with the correct producer code can see the information in the policy file.
	`noPermissionDisplayKey`	No	A display key that provides the text to show if the user does not have a required permission.
`SystemPermType`	`code`	Yes	A code value defined in the `SystemPermissionType` typelist.
`ProducerStatus`	`code`	Yes	The status of the producer associated with the policy. This must be a value defined in the `ProducerStatus` typelist.

The following example illustrates an `<PolicyPeriodProducerCodeHandler>` element that contains multiple producer status codes.

```
<PolicyPeriodProducerCodeHandler permKey="view" desc="Permission to view the policy file"
      isAllowedForPCOfRecord="true">
  <SystemPermType code="viewpolicyfile"/>
  <ProducerStatus code="Limited"/>
  <ProducerStatus code="Suspended"/>
  <ProducerStatus code="Terminating"/>
</PolicyPeriodProducerCodeHandler>
```

Notice that:

- The application permission key is `view`.
- The value of `isAllowedForPCOfRecord` is `true`. Thus, only a producer whose producer code matches the producer code on the policy can see the policy file.
- The system permission type is `viewpolicyfiles`.
- The element explicitly grants permission to view a policy file to producers with a status of `Limited`. PolicyCenter gives any producer with a status of `Active` this permission automatically. Thus, only producers with a status of `Active` or `Limited` can view the details of policy file. As indicated by the `isAllowedForPCOfRecord` element a producer can only view a policy file associated with its producer code.

See also

- “The security configuration file” on page 219
- “Static handler elements” on page 220
- *Application Guide*

part 5

Database administration

chapter 17

Database configuration

This topic discusses database configuration file `database-config.xml` and how to use the file to configure PolicyCenter database options.

See also

- "Database best practices" on page 269
- "Guidewire database direct update policy" on page 271
- "PolicyCenter database back up" on page 272
- "Database consistency checks" on page 272
- "Resize database columns" on page 276
- "Purging unwanted data" on page 276
- "Understanding database statistics" on page 285

Accessing the database configuration file

You access file `database-config.xml` from the Guidewire Studio **Project** window, in the following location:

configuration→config

You can view, but not edit, many of the database configuration parameters from within the PolicyCenter Server Tools:

Info Pages→Database Parameters

See also

- "The database configuration file" on page 237
- "The Database Parameters screen" on page 371

The database configuration file

Guidewire provides the means to configure various aspects of the PolicyCenter database through configuration file `database-config.xml`. This XML-formatted file contains a root `<database>` element with a number of subelements that you use to implement database configuration options specific to your database type.

The `<database>` element has the following syntax. The following code sample shows required attributes in bold font.

```
<database addforeignkeys="true|false" autoupgrade="full|manual" checker="true|false" dbtype="h2|oracle|sqlserver"
env="string"
```

```
    name="string" printcommands="true|false" versionchecksonly="true|false">

<!-- Sets options for the generation of database statistics at the global, database level -->
<databasestatistics databasedegree="integer" incrementalupdatethresholdpercent="integer"
      numappserverthreads="integer" samplingpercentage="integer" useoraclestatspreferences="true|false">

<!-- Sets database statistics options for the named table -->
<tablestatistics action="delete|keep|update" databasedegree="integer" name="string"
      samplingpercentage="integer">

    <!-- Sets database statistics options for the named column on the named table -->
    <histogramstatistics name="string" numbuckets="integer"/>

  </tablestatistics>

 </databasestatistics>

 <!-- Sets options for the connection pool that Guidewire provides -->
<dbcp-connection-pool jdbc-url="string" max-idle="integer" max-total="integer" max-wait="integer"
      min-evictable-idle-time="integer" num-tests-per-eviction-run="integer" password-file="string"
      test-on-borrow="true|false" test-on-return="true|false" test-while-idle="true|false"
      time-between-eviction-runs="integer" when-exhausted-action="block|fail|grow">
  <reset-tools-params collation="string" oracle-tnsnames="string" system-password="string"
        system-username="string"/>
</dbcp-connection-pool>

<!-- Sets the data source for a JBoss, Tomcat, WebLogic, or WebSphere application server-->
<jndi-connection-pool datasource-name="string"/>

<!-- Sets options for an Oracle database -->
<oracle-settings adaptive-optimization="true|false" db-resource-mgr-cancel-sql="string"
      query-rewrite="true|false" statistics-level-all="true|false"
      stored-outline-category="string"/>

 <!-- Sets options for a SQL Server database -->
 <sqlserver-settings jdbc-trace-file="string" jdbc-trace-level="string" unicodecolumns="true|false"/>

 <!-- Sets various options related to database upgrade -->
 <upgrade>

   <mssql-db-ddl>

     <!-- Sets SQL Server database options at the global, database level -->
    <mssql-compression index-compression="NONE|PAGE|ROW" table-compression="NONE|PAGE|ROW/>
    <mssql-filegroups admin="string" index="string" lob="string" op="string" staging="string"
          typelist="string"/>

    <!-- Set SQL Server options for the named table, overrides values set at database level -->
    <mssql-table-ddl table-name="string">
      <mssql-index-ddl filter-where="string"index-compression="NONE|PAGE|ROW"
            index-filegroup="string" key-columns="string" partition-scheme="string"/>
      <mssql-table-compression index-compression="NONE|PAGE|ROW" table-compression="NONE|PAGE|ROW"/>
      <mssql-table-filegroups="string" index-filegroup="string" lob-filegroup
            table-filegroup="string"/>
    </mssql-table-ddl>

   </mssql-db-ddl>

   <ora-db-ddl>

     <!-- Sets Oracle database options at the global, database level -->
    <ora-compression index-compression="true|false" table-compression="ADVANCED|BASIC|NONE"/>
    <ora-lobs caching="true|false" type="BASIC|SECURE|SECURE_COMPRESSED/>
    <tablespaces admin="string" index="string" lob="string" op="string" staging="string"
          typelist="string"/>

     <!-- Sets Oracle options for the named table, overrides values set at the database level -->
    <ora-table-ddl table-name="string">
      <ora-index-ddl index-compression="true|false" index-tablespace="string" key-columns="string"/>
      <ora-lobs caching="true|false" type="BASIC|SECURE|SECURE_COMPRESSED/>
      <ora-table-compression index-compression="true|false" table-compression="ADVANCED|BASIC|NONE">
      <ora-table-date-interval-partitioning datecolumn="string"
            interval="DAILY|MONTHLY|QUARTERLY|WEEKLY|YEARLY">
      <ora-table-hash-partitioning hash-columns="string" num-partitions="integer"/>
```

```
        <ora-table-tablespaces index-tablespace="string" lob-tablespace="string"
              table-tablespace="string"/>
      </ora-table-ddl>

     </ora-db-ddl>

     <versiontriggers dbmsperfinfothreshold="integer">

       <!-- Sets override options for the named database version trigger -->
      <versiontrigger extendedquerytracingenabled="true|false" name="string"
            parallel-dml="true|false" parallel-query="true|false"
            queryoptimizertracingenabled="true|false" recordcounters="true|false"
            updatejoinorderedhint="true|false" updatejoinusemergehint="true|false"
            updatejoinusenlhint="true|false"/>

     </versiontriggers>

   </upgrade>

 </database>
```

File `database-config.xml` contains a single root-level `<database>` element that takes the following attributes.

`name`	*Required*. String identifying the database for which PolicyCenter uses this connection specification.
`dbtype`	*Required*. Database type, either `h2` (for the QuickStart database), `oracle`, or `sqlserver`.
The following attributes are all optional.	
`addforeignkey`	Used only for development and testing. Do not use this attribute in production. The default is `true`.
`autoupgrade`	Use to set how to upgrade the database. Valid values are: • `full` – Takes precedence and initiates a full upgrade assuming all other necessary conditions are met. • `manual` – Requires that you set either the database upgrade type (in Server Tools Upgrade and Versions screen) or the date system property.
`checker`	*Boolean*. Whether PolicyCenter runs consistency checks before it starts: • Development environments – For development environments with small data sets, you can enable consistency checks to run each time the PolicyCenter server starts. Set the value of `checker` in the database block to `true` to enable checks on server startup. • Production environments – Running consistency checks upon server startup can take a long time, impact performance severely, and possibly time out on very large datasets. Set the value of `checker` in the database block to `false` to disable checks on server startup. Valid values are: • `true` – Guidewire recommends that you only set `checker` to `true` in development environments with a small set of test data. • `false` – Guidewire recommends that you set `checker` to `false` under most circumstances. The default is `true`. See the following for more information: • “Database consistency checks” on page 272 • “Configure consistency checks to run at server startup” on page 274
`env`	Use of the `env` attribute to set a server environment enables you to provide different database configurations for different server environments. For example, you can set up different database configurations for a production environment and a test environment. To specify a database configuration for multiple environments, provide a comma-separated list of values for the `env` attribute. See “Example syntax for the <server> element” on page 62 for more information.
`printcommands`	*Boolean*. Whether the server prints database upgrade messages to the console upon startup. Valid values are: • `true` - By default, Guidewire sets the value of `printcommands` to `true` in the base configuration. • `false` - Do not set `printcommands` to `false` in a production environment. The default is `true`.

`versionchecksonly`	*Boolean*. Whether the PolicyCenter server runs only database version checks at startup, without performing any actual database upgrade steps: • `true` - PolicyCenter runs all version checks regardless of a failure in one of the checks. • `false` - PolicyCenter stops the upgrade if it detects an error. The default is `false`. Changes to this attribute take effect only during an application upgrade.

The `<database>` element takes the following subelements. There is, at most, a single occurrence of each of these subelements in the `<database>` element.

`databasestatistics`	Specifies parameters that control the generation of database statistics. See "The <databasestatistics> database configuration element" on page 241 and "Database statistics generation" on page 285 for more information.
`dbcp-connection-pool`	Specifies parameters for connection pool shared using `dbcp`. You must include this subelement if using a `dbcp` data source. See "The <dbcp-connection-pool> database configuration element" on page 242 and the *Installation Guide* for more information.
`jndi-connection-pool`	Specifies parameters for a connection pool shared using JNDI. You must include this subelement if using a `jndi` data source. See "The <jndi-connection-pool> database configuration element" on page 245 and the *Installation Guide* for more information.
`oracle-settings`	Specifies settings for Oracle databases. See "The <oracle-settings> database configuration element" on page 248 and the *Installation Guide* for more information.
`sqlserver-settings`	Specifies settings for SQL Server databases. See "The <sqlserver-settings> database configuration element" on page 249 and the *Installation Guide* for more information.
`upgrade`	Specifies PolicyCenter behavior during a database upgrade. See "The <upgrade> database configuration element" on page 250 for more information.

See also

- *Installation Guide*

The database autoupgrade attribute

You use the `autoupgrade` attribute on the `<database>` element in file `database-config.xml` to manage PolicyCenter configuration and database upgrades. The `autoupgrade` attribute has the following syntax:

```
<database ... autoupgrade="full|manual" .../>
```

The attribute value has the following meaning. The default value is `manual` if the attribute is missing.

autoupgrade attribute	**Description**
`full`	Whenever the application server starts, if it determines the need for a full upgrade, the presence of this attribute set to `full` is sufficient permission to perform the upgrade. With this setting: • If a rolling (configuration) upgrade is already in progress as the server starts, the server throws an exception, to force the choice of an upgrade type. • If a full upgrade is already in progress by other means as the server starts, there is no issue as this setting is consistent with the a full upgrade.
`manual`	This setting requires that you explicitly set the permission to upgrade through one of the following means: • Setting the database upgrade type in the Server Tools **Upgrade and Versions** screen and initiating the upgrade from that screen. • Setting the following Java system property to the current date as the application server starts: `-Dgw.pc.full.upgrade.intended.date` See "Unexpected upgrades" on page 208 for a discussion of the use of this Java system property. You must set the value of `autoupgrade` to `manual` if performing a rolling (configuration) upgrade.

The behavior of this attribute depends on the following:

- Whether you are working with a single PolicyCenter server or with multiple PolicyCenter servers in a clustered server environment.
- Whether you are attempting a full PolicyCenter database upgrade or using a rolling upgrade to implement application configuration changes.

Single PolicyCenter server installations

The following tables describe the interactions between setting **Start Full Upgrade** on the **Upgrade and Versions** screen and the value of the `autoupgrade` attribute.

Start Full Upgrade	autoupgrade	Result
Set	Not set or `full`	The server starts, the upgrade completes, and PolicyCenter updates the **Upgrade and Versions** screen.
Not set	Not set or `manual`	The server does not start, the upgrade fails, and PolicyCenter logs an error message.

The following table describes the interactions between setting **Start Full Upgrade** on the **Upgrade and Versions** screen and the value of the `autoupgrade` attribute during deployment of non-data model changes to a production mode server.

Start Full Upgrade	autoupgrade	Result
Set	Not set or `full`	The server starts, the upgrade completes, and PolicyCenter updates the **Upgrade and Versions** screen.
Not set	Not set or `manual`	The PolicyCenter server starts.

Clustered PolicyCenter server installations

In clustered PolicyCenter server installations, the `autoupgrade` attribute has the following behavior.

autoupgrade	Result
`full`	If you set the value of `autoupgrade` attribute to `full`, any attempt to start a rolling upgrade fails. In addition, you must always set the upgrade type (full) using the Server Tools **Upgrade and Versions** screen, or, by using the `system_tools -startfullupgrade` command option, for example.
`manual`	If you set the value of `autoupgrade` attribute to `manual`, you must always set the upgrade type (full or rolling) or the upgrade fails. You can set the upgrade type in the Server Tools **Upgrade and Versions** screen or by setting a JVM parameter at server startup. For a rolling upgrade, the new configuration (`database-config.xml`) must set this value to `manual`. This new value overrides any value set in the old configuration.
Not set	If you do not set the value of `autoupgrade` attribute, PolicyCenter assumes a default value of `manual` and behaves accordingly.

The <databasestatistics> database configuration element

The `<database>` element in file `database-config.xml` contains a single occurrence of subelement `<databasestatistics>`. Use this element to control how PolicyCenter generates database statistics statements.

The `<databasestatistics>` element has the following syntax. The following code sample shows required attributes in bold font.

```
<database>

   <databasestatistics databasedegree="integer" incrementalupdatethresholdpercent="integer"
        numappserverthreads="integer" samplingpercentage="integer"
        useoraclestatspreferences="true|false">

    <!-- Sets database statistics options for the named table -->
    <tablestatistics action="delete|keep|update|force" databasedegree="integer" name="string"
          samplingpercentage="integer">

    <!-- Sets database statistics options for the named column on the named table -->
    <histogramstatistics name="string" numbbuckets="integer" />

     </tablestatistics>

   </databasestatistics>

 </database>
```

The following list describes the attributes that you can configure on the `<databasestatistics>` element. All of these attributes are optional. See “The <databasestatistics> database configuration element” on page 291 for more information on these attributes.

`databasedegree`	On Oracle, this attribute controls the degree of database parallelism that Oracle uses in executing each individual statement. The default is 1. PolicyCenter uses the value of this attribute for all statements. SQL Server ignores the `databasedegree` attribute.
`incrementalupdatethresholdpercent`	This attribute 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 PolicyCenter uses to update database statistics for staging tables during import only.
`samplingpercentage`	The behavior of this attribute depends on the database type. For Oracle, Guidewire recommends that you always set this value to 0 to enable Oracle auto-sampling.
`useoraclestatspreferences`	On Oracle, this attribute sets the database statistics preferences to be able to use the Oracle Autotask infrastructure instead of the `DBStats` batch process from PolicyCenter. The default is `false`, which requires that you disable the Autotask and schedule `DBStats` batch processing in its place. Changes to the value of this attribute only take effect during an application upgrade.

The `<databasestatistics>` element has the following subelement.

`tablestatistics`	Provides overrides of database-wide statistics settings defined on the `<databasestatistics>` element for a specific table. There can be multiple occurrences of the `<tablestatistics>` subelement on the `<databasestistics>` element.

See also

- “The database configuration file” on page 237
- “Understanding database statistics” on page 285
- “Configuring database statistics generation” on page 290
- “The <databasestatistics> database configuration element” on page 291
- “The <tablestatistics> database configuration element” on page 294
- “The Database Statistics screen” on page 375
- “Using Oracle AutoTask for statistics generation” on page 292
- “system_tools command options” on page 429

The <dbcp-connection-pool> database configuration element

The `<database>` element in file `database-config.xml` contains, at most, a single occurrence of subelement `<dbcp-connection-pool>`. The use of the `<dbcp-connection-pool>` element is optional. If using PolicyCenter to manage the connection pool rather than a JNDI data source managed by the application server, you must use this element to configure the connection pool behavior.

> **Note:** Guidewire implements its own version of the Apache Commons Pool, overriding specific values to provide improved functionality.

If you experience slow performance, it is possible 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 using a simple query and evicts idle connections and those that fail with an exception.

The `<dbcp-connection-pool>` element has the following syntax. The following code sample shows required attributes in bold font.

```
<database>

   <!-- Sets options for the connection pool that Guidewire provides -->
  <dbcp-connection-pool jdbc-url="string" max-idle="integer" max-total="integer" max-wait="integer"
        min-evictable-idle-time="integer" num-tests-per-eviction-run="integer" password-file="string"
        test-on-borrow="true|false" test-on-return="true|false" test-while-idle="true|false"
        time-between-eviction-runs="integer" when-exhausted-action="block|fail|grow">
    <reset-tools-params collation="string" oracle-tnsnames="string" system-password="string"
          system-username="string"/>
  </dbcp-connection-pool>

 </database>
```

The following list describes the attributes that you can configure on the `<dbcp-connection-pool>` element.

> **Note:** These attributes apply only if you use the default connection pool. If you use the server connection pool, these settings do not apply. Configure the server connection pool instead through the administration console provided with the server. See the *Installation Guide* for more information.

`jdbc-url`	*Required*. Stores connection information for the database. The format of the `jdbc-url` value changes depending on the database type. See the *Installation Guide* for more information.
The following attributes are all optional.	
`max-idle`	Maximum number of connections that can sit idle in the pool at any time. If negative, there is no limit to the number of connections that can be idle at any given time. The default is -1.
`max-total`	Maximum number of connections that the connection pool can allocate, including those in use by a client or that are in an idle state awaiting use. A reasonable initial value for this is about 25% of the number of users that you expect to use PolicyCenter at the same time. If set to a negative integer, there is no limit to the number of allowed database connections. The default is -1. If the number of database connections reaches the value of `max-total`, PolicyCenter considers the connection pool to have no more available connections.
`max-wait`	Maximum amount of time, in milliseconds, that the data source waits for a connection before one becomes available in the pool to service. The default is 30000. The value of the `max-wait` attribute interacts with the `when-exhausted-action` attribute. See that attribute for more information.
`min-evictable-idle-time`	Maximum time, in milliseconds, that a connection can sit idle in the pool before it is eligible for eviction due to idle time. If a connection is idle more the specified number of milliseconds, PolicyCenter evicts the connection from the pool. The default is 300000 milliseconds. If this value is a non-positive integer, PolicyCenter does not drop connections from the pool due to idle time alone. This setting has no effect unless the value of `time-between-eviction-runs` is greater than 0.

	During an eviction run, PolicyCenter scans the connection pool and tests the number of idle connections equal to the value of `num-tests-per-eviction-run`.
`num-tests-per-eviction-run`	Number of idle connections that PolicyCenter tests in each eviction run. This setting has no effect unless the value of `time-between-eviction-runs` is greater than `0`. The default is 3.
`password-file`	Use to hide the value of the database connection password in the `jdbc-url` connection string. Instead of providing the password in the connection string, you can place the password in an external file and reference this file from file `database-config.xml`. See the *Installation Guide* for more information.
`test-on-borrow`	*Boolean*. Whether PolicyCenter tests a connection by running a simple validation query as PolicyCenter first borrows the connection from the connection pool. If set to `true`, the connection pool attempts to validate each connection before PolicyCenter uses the connection from the connection pool. If a connection fails validation, the connection pool drops the connection and chooses a different connection to borrow. The default is `false`. PolicyCenter returns any connection used only for a query to the pool immediately after the query completes. Thus, running a test query every time that a connection returns to the pool can potentially affect performance.
`test-on-return`	*Boolean*. Whether PolicyCenter tests a connection by running a simple validation query as PolicyCenter returns the connection to the connection pool. If set to `true`, the connection pool attempts to validate each connection that PolicyCenter returns from the database. The default is `false`. PolicyCenter returns any connection used only for a query to the pool immediately after the query completes. Thus, running a test query every time that a connection returns to the pool can potentially affect performance.
`test-while-idle`	*Boolean*. Whether PolicyCenter performs validation on idle connections in the connection pool. If set to `true`, the connection pool performs validation on idle connections. It drops connections that fail the validation test. The default is `true`. This attribute value has no effect unless the value of `time-between-eviction-runs` is greater than zero.
`time-between-eviction-runs`	Time, in milliseconds, that PolicyCenter waits between eviction runs of idle connections in the connection pool. The default is 60000. If set to a a non-positive integer, PolicyCenter does not launch any eviction threads.
`when-exhausted-action`	Specifies the behavior of the connection pool if the pool has no more connections. Set this attribute to one of the following values: • `fail` – If the there are no more connections available, PolicyCenter throws a `NoSuchElementException` exception. • `grow` – If there are no more connections available, PolicyCenter creates a new connection and returns it, essentially making `max-active` meaningless. • `block` – If there are no more connections available, PolicyCenter blocks connections until a new or idle connection becomes available. If the value of `max-wait` is positive, then PolicyCenter blocks, at most, for that number of milliseconds, after which PolicyCenter throws a `NoSuchElementException` exception. If the value of `max-wait` is non-positive, PolicyCenter blocks indefinitely. The default is `block`.

The `<dbcp-connection-pool>` element has the following subelement.

`reset-tools-params`	See “The <reset-tool-params> database configuration element” on page 245 for more information.

See also

- “The database configuration file” on page 237
- “The Database Parameters screen” on page 371

The <reset-tool-params> database configuration element

The `<dbcp-connection-pool>` element in file `database-config.xml` contains, at most, a single occurrence of subelement `<reset-tool-params>`. The use of the `<reset-tool-params>` element is optional. Use this element to configure `DBResetTool`, not the DBCP connection pool. Guidewire defines the `<reset-tool-params>` element as a subelement of the `<dbcp-connection-pool>` element because `DBResetTool` only works on databases defined in the `<dbcp-connection-pool>` element.

The `<reset-tool-params>` element has the following syntax.

```
<database>
  <dbcp-connection-pool>
    <reset-tools-params collation="string" oracle-tnsnames="string" system-password="string"
          system-username="string"/>
  </dbcp-connection-pool>
</database>
```

The following list describes the attributes that you can configure on the `<reset-tools-params>` element. All of these attributes are optional.

`collation`	Collation value to use if creating a new H2 (QuickStart) or SQL Server database: • H2 – Sets database collation using the Java Collation class. • SQL Server – Sets database collation to a Microsoft Window’s or SQL collation name. `DBResetTool (dropdb)` uses the value of this attribute if creating a new H2 or SQL Server database.
`oracle-tnsnames`	(Oracle) Name of the Oracle `tnsnames.ora` file.
`system-password`	(Oracle) Database system password.
`system-username`	(Oracle) Database system username.

The `<reset-tools-arams>` element has no subelements.

See also

- “The database configuration file” on page 237

The <jndi-connection-pool> database configuration element

The `<database>` element in file `database-config.xml` contains, at most, a single occurrence of subelement `<jndi-connection-pool>`. The use of the `<jndi-connection-pool>` element is optional. However, file `database-config.xml` must contain this element if you use a JNDI (Java Naming and Directory Interface) data source managed by a JBoss, Tomcat, WebLogic, or WebSphere application server.

The `<jndi-connection-pool>` element has the following syntax. The following code sample shows required attributes in bold font.

```
<database>

  <!-- Sets the data source for a JBoss, Tomcat, WebLogic, or WebSphere application server-->
  <jndi-connection-pool datasource-name="string"
        connections-initialized-for-application="true|false"/>

</database>
```

IMPORTANT If you modify the `<jndi-connection-pool>` element in any way, you must restart the application server.

The following list describes the attributes that you can configure on the `<jndi-connection-pool>` element.

`datasource-name`	*Required*. Specifies the JNDI name to assign to the data source. See the *Installation Guide* for more information.
The following attribute is optional.	
`connections-initialized-for-application`	(Oracle) *Boolean*. Controls the number of SQL statements that PolicyCenter executes on every connection that it borrows from an external data source. This setting applies to the named JNDI database connection set up in this `<jndi-connection-pool>` element only. Valid values are: • `true` – If configured appropriately, the data source provides connections with certain Oracle database parameters set to their desired values. • `false` – PolicyCenter runs a set of SQL statements on each and every database connection that it borrows from the data source. The default is `false`. To take advantage of this feature: • Your PolicyCenter installation must use an Oracle database. • You must configure the data source appropriately. See “Configuring JNDI connection initialization for Oracle” on page 246 for more information. **IMPORTANT** If you set this attribute to `true` and do not initialize the connections properly, the application server refuses to start and logs an error message for each incorrect setting.

The `<jndi-connection-pool>` element has no subelements.

See also

- “The database configuration file” on page 237
- “Configuring JNDI connection initialization for Oracle” on page 246

Configuring JNDI connection initialization for Oracle

Boolean attribute `connections-initialized-for-application` on the `<jndi-connection-pool>` element in `database-config.xml` configures how the JNDI data source provides connections to PolicyCenter.

Attribute connections-initialized-for-application set to true

If you set the value of attribute `connections-initialized-for-application` to `true`, you must provide the correct settings for the Oracle database parameters that the external data sources uses for connection initialization. If you do not initialize the connections properly, the application server refuses to start and logs an error message for each incorrect setting. Guidewire recommends that you set this attribute to `true`, then start the application server. You can then determine the correct values to use for your database from the log error messages.

In general, the Oracle database parameters to use for connection initialization have the following meanings:

Oracle parameter	Meaning
`CURSOR_SHARING`	*Boolean*. The exact setting is dependant on your database installation.

Oracle parameter	Meaning
`MODULE`	The module name, which must include the database user name as well. For example, if your database user name is PCPROD1, then set the module name to PolicyCenter_PCPROD1.
`NLS_SORT`	The default sort collation for the Oracle database. The exact setting is dependent on your Guidewire installation. See the *Globalization Guide* for more information.
`NLS_COMP`	The database collation behavior. Set this value to `BINARY`.

To use this feature, you must configure the application server to manage the connection initialization. See “Set Oracle database parameters for connection initialization” on page 247 for more information.

Attribute connections-initialized-for-application set to false

If you set the value of attribute `connections-initialized-for-application` to `false`, the external data source takes no action to configure the borrowed connections before placing an item in the connection pool. This causes PolicyCenter to run a set of SQL statements on each and every connection that it borrows from the data source.

Although the cost in time of each of these statements is very small, Guidewire applications sometimes borrow connections at a very high rate. Thus, it is possible for the time to execute these statements to become measurable and to become visible during performance analysis in the Guidewire Profiler.

Set Oracle database parameters for connection initialization

About this task

Boolean attribute `connections-initialized-for-application` on the `<jndi-connection-pool>` element in `database-config.xml` configures how the JNDI data source provides connections to PolicyCenter. To use this feature with an external data source, you must configure your data source to initialize a set of Oracle database parameters.

Procedure

1. Set attribute `connections-initialized-for-application` on the `<jndi-connection-pool>` element to `true`.
2. Start the application server.
3. From the server log, determine the exact values to set for connection initialization.
4. Depending on your application server, set up the connection initialization as appropriate.
   See “Connection initialization for Oracle databases” on page 247 for more information.
5. After completing your initialization configuration, restart the application server and Guidewire PolicyCenter.

Connection initialization for Oracle databases

If using an Oracle database, it is possible to configure an application server to initialize a data pool connection before PolicyCenter uses that connection. This can improve performance. To use this functionality, you must set Boolean attribute `connections-initialized-for-application` on the `<jndi-connection-pool>` element in `database-config.xml` to `true`.

WebLogic application servers

Use the WebLogic **Administration Console** to specify a SQL statement that initializes each data source connection before WebLogic adds that connection to the pool.

If you have more than one statement to execute, you can put the statements into a SQL block. For example:

```
SQL BEGIN
  DBMS_APPLICATION_INFO.SET_MODULE('PolicyCenter_PCPROD1', NULL);
  EXECUTE IMMEDIATE 'ALTER SESSION SET NLS_SORT = BINARY_CI';
END;
```

> **Note:** In the sample code, replace `PolicyCenter_PCPROD1` with the actual name of the Oracle database followed by the logon user username.

See also

- "Configuring JNDI connection initialization for Oracle" on page 246

Non-weblogic application servers

One way to initialize the connections appropriately is to create an Oracle logon trigger for the owner of the database schema. The following sample code illustrates how to create a logon trigger. You must be logged on as the `system` user to execute this trigger.

```
CREATE OR REPLACE TRIGGER
AFTER LOGON ON DATABASE
BEGIN

 IF (USER = 'Guidewire schema owner')
    THEN
      DBMS_APPLICATION_INFO.SET_MODULE('PolicyCenter_PCPROD1', NULL);
      EXECUTE IMMEDIATE  'ALTER SESSION SET NLS_SORT = BINARY_CI';
END IF;

 END;
/
```

> **Note:** In the sample code, replace `Guidewire schema owner` with Oracle database system username. Replace `PolicyCenter_PCPROD1` with the actual name of the Oracle database followed by the logon user username.

See also

- "Configuring JNDI connection initialization for Oracle" on page 246

The <oracle-settings> database configuration element

The `<database>` element in file `database-config.xml` contains, at most, a single occurrence of subelement `<oracle-settings>`. The use of the `<oracle-settings>` element is optional. Use this element to configure Oracle-only database parameters.

The `<oracle-settings>` element has the following syntax.

```
<database>

   <oracle-settings adaptive-optimization="true|false" db-resource-mgr-cancel-sql="string"
        query-rewrite="true|false" statistics-level-all="true|false"
        stored-outline-category="string"/>

</database>
```

The following list describes the attributes that you can configure on the `<oracle-settings>` element. All of these attributes are optional.

`adaptive-optimization`	Specifies the behavior of the Oracle Adaptive Optimization feature. Valid values are: • `true` - Does nothing; default if not set • `false` - Sets the parameter `OPTIMIZER_ADAPTIVE_PLANS` to `false` at the session level
`db-resource-mgr-cancel-sql`	Name of an Oracle Resource Consumer Group, if any.
`query-rewrite`	*Boolean*. Whether to enable query rewrite.

	Valid values are: • `true` – Enable query rewrite and use a matching materialized view. • `false` – Set the Oracle `query-rewrite` parameter to `false` to disable use of Oracle materialized views. If not present, Guidewire does not set this value at the session level
`statistics-level-all`	*Boolean*. Whether to set the Oracle `statistics_level` parameter to `ALL` and enable collection of detailed execution plan statistics. The default is `false`.
`stored-outline-category`	Name of the stored outline category to use, if any

The `<oracle-settings>` element does not contain additional subelements.

See also

- "The database configuration file" on page 237

The <sqlserver-settings> database configuration element

The `<database>` element in file `database-config.xml` contains, at most, a single occurrence of subelement `<sqlserver-settings>`. The use of the `<sqlserver-settings>` element is optional. Use this element to configure Microsoft JDBC driver logging.

The `<sqlserver-settings>` element has the following syntax.

```
<database>
  ...
  <sqlserver-settings jdbc-trace-file="string" jdbc-trace-level="string"
        unicodecolumns="true|false"/>
  ...
</database>
```

The following list describes the attributes that you can configure on the `<sqlserver-settings>` element. All of these attributes are optional.

`jdbc-trace-file`	Specifies the name of the trace file. If you do not provide a file name, this value defaults to the following:
	`C:\temp\msjdbctrace%u.log`
	PolicyCenter replaces the symbols in the file name at runtime with their meaning as listed at the following web site.
	`http://java.sun.com/j2se/1.5.0/docs/api/java/util/logging/FileHandler.html`
	Use the listed symbols to uniquely name the trace file.

`jdbc-trace-level`	Valid trace level as listed at the following web site: `http://msdn.microsoft.com/en-us/library/ms378517(SQL.90).aspx?ppud=4`
`unicodecolumns`	Required if starting a new database that exclusively uses Unicode-capable column character data types (`nvarchar`, ...). PolicyCenter ignores this attribute if the database does not support Unicode or if the attribute is not relevant to the new database. The default is `false`.

The `<sqlserver-settings>` element does not contain additional subelements.

See also

- "The database configuration file" on page 237
- *Installation Guide*

The <upgrade> database configuration element

The `<database>` element in file `database-config.xml` contains, at most, a single occurrence of subelement `<upgrade>`. The use of the `<upgrade>` element is optional. This element specifies various options related to database upgrade. One important area of configuration is the degree of database parallelism to use in an Oracle database. Database parallelism refers to the ability of an Oracle database to execute a database SQL statement such as `CREATE` or `INSERT` using simultaneous, parallel slave processes.

The `<upgrade>` element has the following syntax. The following code sample shows required attributes in bold font.

```
<database>
  <upgrade>

     <mssql-db-ddl>

       <!-- Sets SQL Server database options at the global, database level -->
      <mssql-compression index-compression="NONE|PAGE|ROW" table-compression="NONE|PAGE|ROW/>
      <mssql-filegroups admin="string" index="string" lob="string" op="string" staging="string"
            typelist="string"/>

      <!-- Set SQL Server options for the named table, overrides values set at database level -->
      <mssql-table-ddl table-name="string">
        <mssql-index-ddl filter-where="string"index-compression="NONE|PAGE|ROW"
              index-filegroup="string" key-columns="string" partition-scheme="string"/>
        <mssql-table-compression index-compression="NONE|PAGE|ROW" table-compression="NONE|PAGE|ROW"/>
        <mssql-table-filegroups="string" index-filegroup="string" lob-filegroup
              table-filegroup="string"/>
      </mssql-table-ddl>

     </mssql-db-ddl>

     <ora-db-ddl>

       <!-- Sets Oracle database options at the global, database level -->
      <ora-compression index-compression="true|false" table-compression="ADVANCED|BASIC|NONE"/>
      <ora-lobs caching="true|false" type="BASIC|SECURE|SECURE_COMPRESSED/>
      <tablespaces admin="string" index="string" lob="string" op="string" staging="string"
            typelist="string"/>

       <!-- Sets Oracle options for the named table, overrides values set at the database level -->
      <ora-table-ddl table-name="string">
        <ora-index-ddl index-compression="true|false" index-tablespace="string" key-columns="string"/>
        <ora-lobs caching="true|false" type="BASIC|SECURE|SECURE_COMPRESSED/>>
        <ora-table-compression index-compression="true|false" table-compression="ADVANCED|BASIC|NONE">
        <ora-table-date-interval-partitioning datecolumn="string"
              interval="DAILY|MONTHLY|QUARTERLY|WEEKLY|YEARLY">
        <ora-table-hash-partitioning hash-columns="string" num-partitions="integer"/>
        <ora-table-tablespaces index-tablespace="string" lob-tablespace="string"
              table-tablespace="string"/>
      </ora-table-ddl>

     </ora-db-ddl>

     <versiontriggers dbmsperfinfothreshold="integer">
      <!-- Sets override options for the named database version trigger -->
      <versiontrigger extendedquerytracingenabled="true|false" name="string"
            parallel-dml="true|false" parallel-query="true|false"
            queryoptimizertracingenabled="true|false" recordcounters="true|false"
            updatejoinorderedhint="true|false" updatejoinusemergehint="true|false"
            updatejoinusenlhint="true|false"/>
    </versiontriggers>

   </upgrade>
</database>
```

The following list describes the attributes that you can configure on the `<upgrade>` element. All of these attributes are optional.

`allowUnloggedOperations`	*Boolean*. Whether to disable logging of certain SQL operations during the database upgrade. Valid values are: • `true` – Run the upgrade with minimal database redo logging and enable direct-path `INSERT` operations. • `false` – Run the upgrade with standard database redo logging. The default is `false`. **Note:** If you run the upgrade with attribute `allowUnloggedOperations` set to `true`, then you need to take a full database backup after the upgrade.
`collectstorageinstrumentation`	(Oracle) *Boolean*. Whether PolicyCenter collects tablespace usage and object size data before and after the upgrade. Valid values are: • `true` – PolicyCenter collects tablespace usage and size of segments such as tables, indexes and LOBs (large object binaries) before and after the upgrade. You can then compare the before and after values to find the utilization change caused by the upgrade. • `false` – PolicyCenter does not collect this data. The default is `false`.
`defer-create-nonessential-indexes`	*Boolean*. Whether to defer creation of non-essential indexes during the upgrade process until the upgrade completes and the application server is back up. Creation of non-essential indexes can add significant time to the upgrade duration. Valid values are: • `true` – Defer creation of non-essential indexes during upgrade. • `false` – Do not defer creation of non-essential indexes during upgrade. The default is `false`. Non-essential indexes are: • Performance-related indexes that do not enforce constraints. • Indexes on the `ArchivePartition` column on all entities that PolicyCenter can archive. If you choose to defer creation of non-essential indexes, PolicyCenter runs the Deferred Upgrade Tasks batch process (`DeferredUpgradeTasks`) as soon as the upgrade completes and the server starts up. See “Deferred Upgrade Tasks batch process” on page 139 for more information.
`deferDropColumns`	(Oracle) *Boolean*. Whether to drop table columns removed during upgrade immediately or leave their removal to a later time. The database upgrade removes some columns. For Oracle, you can configure whether the removed columns are dropped immediately or are marked as unused. Marking a column as unused is a faster operation than dropping the column immediately. However, as PolicyCenter does not physically drop the removed columns from the database, the space used by these columns is not released immediately to the table and index segments. Valid values are: • `true` – Defer dropping removed columns until after the upgrade, possibly during off-peak hours of operation. The PolicyCenter database upgrade marks the removed columns as unused instead. • `false` – Drop the removed columns immediately, during the upgrade process. The default is `true`.
`degree-of-parallelism`	(Oracle) Controls the degree of database parallelism that Oracle uses for `INSERT`, `UPDATE`, and `DELETE` database operations. Valid values are: • 0 – Defers to Oracle to determine the degree of database parallelism for the operations that the attribute configures. The Oracle automatic parallel tuning feature determines the degree based on the number of CPU processors

	involved and the value set for the Oracle parameter `PARALLEL_THREADS_PER_CPU`. • 1 – Disables the parallel execution of DDL statements. • Positive integer less than 1000 – Database parallelism, with the specified value as the degree of parallelism. The default is 4.
`degree-parallel-ddl`	(Oracle) Controls the degree of database parallelism that Oracle uses to execute DDL (Data Definition Language) statements during the database upgrade. Use to configure the degree of database parallelism for commands such as `CREATE INDEX` and the `ALTER TABLE` commands. Valid values are: • 0 – Defers to Oracle to determine to determine the degree of database parallelism for the operations that the attribute configures. The Oracle automatic parallel tuning feature determines the degree based on the number of CPUs involved and the value set for the Oracle parameter `PARALLEL_THREADS_PER_CPU`. • 1 – Disables the parallel execution of DDL statements. • Positive integer less than 1000 – Database parallelism, with the specified value as the degree of parallelism. The default is 4. If you set the value of `ora-parallel-dml` to `enable` or `enable_all` (default), then you need to provide a value for attribute `degree-of-parallelism` as well.
`encryptioncommitsize`	Sets the commit size for rows requiring encryption. If one or more attributes use PolicyCenter encryption, the PolicyCenter database upgrade commits batches of encrypted values. The upgrade commits `encryptioncommitsize` rows at a time in each batch. The default value of `encryptioncommitsize` varies based on the database type: • Oracle – 10000 • SQL Server – 100 Test the upgrade on a copy of your production database before attempting to upgrade the actual production database. If the encryption process is slow, and you cannot attribute the slowness to SQL statements in the database, try adjusting the `encryptioncommitsize` attribute. After you optimized the performance of the encryption process, use that value of `encryptioncommitsize` as you upgrade your production database.
`ora-parallel-dml`	(Oracle) Controls database parallelism usage by Oracle in the execution of DML (Data Manipulation Language) operations. Valid values are: • `disable` – Oracle does not execute DML statements in parallel during upgrade. • `enable` – Oracle executes DML statements in parallel during upgrade, if configured to do so. • `enable_all` – Oracle executes DML statements in parallel during upgrade in all cases, unless turned off in the code or through configuration. The default is `enable_all`. If you set the value of `ora-parallel-dml` to `enable` or `enable_all`, then you need to provide a value for attribute `degree-of-parallelism` as well. **Note:** The value of this attribute interacts with the `parallel-dml` attribute on the `<versiontrigger>` element. See “The <versiontrigger> database configuration element” on page 267 for more information.
`ora-parallel-query`	(Oracle) Controls parallel query usage by Oracle during a database upgrade. Valid values are: • `disable` – Oracle does not use parallel queries during upgrade. • `enable` – Oracle uses parallel queries during upgrade, if configured to do so. The default is `enable`.

	The value of this attribute interacts with the `parallel-query` attribute on the `<versiontrigger>` element. See "The <versiontrigger> database configuration element" on page 267 for more information.
`sqlserverCreateIndexSortInTempDB`	(SQL Server) *Boolean*. Whether SQL Server stores temporary sort results in tempdb. By using tempdb for sort runs, disk input and output is typically faster, and the created indexes tend to be more contiguous. Valid values are: • `true` – SQL Server stores sort results in tempdb. • `false` – SQL Server stores sort results in the destination filegroup. The default is `false`. If you set `sqlserverCreateIndexSortInTempDB` to `true`, you must have enough disk space available to tempdb for the sort runs, which, for the clustered index, includes the data pages. You must also have sufficient free space in the destination filegroup to store the final index structure, because SQL Server creates the new index before deleting the old index. Refer to the following web site for details on the requirements to use tempdb for sort results. `http://msdn.microsoft.com/en-us/library/ms188281.aspx`
`updatestatistics`	(Oracle) *Boolean*. Whether to update table statistics during upgrade. The overall time that it takes to upgrade the database is shorter if the database upgrade does not update statistics. Valid values are: • `true` – Enables the upgrader to update statistics on changed objects. It also allows the upgrader to maintain column level statistics consistent with what is allowed in the code, data model, and configuration. • `false` – Disable statistics generation during the upgrade. If PolicyCenter does not update statistics during the upgrade: • It reports a warning that recommends that you run the database statistics batch process (`DBStats`) in incremental mode during the next maintenance window. • It updates the Server Tools **Upgrade and Versions** screen to show that the upgrade did not update statistics. If PolicyCenter does generate statistics during the upgrade, it updates the **Upgrade and Versions** screen to report the runs of the statistics batch process, including incremental runs. **Note:** Guidewire recommends that you run statistics in full mode after an upgrade to a major PolicyCenter version. See the following for more information: • "Database Statistics work queue" on page 138 • "Configuring database statistics generation" on page 290 • "The Upgrade and Versions screen" on page 397
`verifyschema`	*Boolean*. Whether PolicyCenter performs a verification of the database schema before starting the database upgrade. This process verifies that the PolicyCenter data model matches the physical database. This process can take some time. The default is `true`. It is possible for the verification process to take some time. Guidewire recommends that you perform this verification prior to starting the upgrade through the use of the `system_tools -verifydbschema` command option. To use the command, enter the following at a command prompt: `system_tools -password` *`password`* `-verifydbschema` See "system_tools command" on page 429 for more information.

The `<upgrade>` element has the following subelements. Each of these elements is optional. There is, at most, a single occurrence of each of these subelements on the `<upgrade>` element.

`mssql-db-ddl`	Specifies options for SQL Server database DDL (Data Definition Language) statements. See "The <mssql-db-ddl> database configuration element" on page 254 for details.

`ora-db-ddl`	Specifies options for Oracle database DDL (Data Definition Language) statements. See “The <ora-db-ddl> database configuration element” on page 259 for details.
`versiontriggers`	Specifies options for named version triggers. See “The <versiontriggers> database configuration element” on page 266 for details.

See also

- “The database configuration file” on page 237

The <mssql-db-ddl> database configuration element

The `<upgrade>` element in file `database-config.xml` contains, at most, a single occurrence of subelement `<mssql-db-ddl>`. The use of the `<mssql-db-ddl>` element is optional. Use this element to set SQL Server database DDL (Data Definition Language) options during the creation of new objects in the database. This configuration applies to the database at a global level.

The `<mssql-db-ddl>` element has the following syntax. The following code sample shows required attributes in bold font.

```
<database>
  <upgrade>

     <mssql-db-ddl>
      <mssql-compression index-compression="NONE|PAGE|ROW" table-compression="NONE|PAGE|ROW/>
      <mssql-filegroups op="string" admin="string" typelist="string" staging="string"
           index="string" lob="string"/>
      <mssql-table-ddl table-name="string">
        <mssql-index-ddl filter-where="string"index-compression="NONE|PAGE|ROW"
              index-filegroup="string" key-columns="string" partition-scheme="string"/>
        <mssql-table-compression index-compression="NONE|PAGE|ROW" table-compression="NONE|PAGE|ROW"/>
        <mssql-table-filegroups="string" index-filegroup="string" lob-filegroup
              table-filegroup="string"/>
      </mssql-table-ddl>
    </mssql-db-ddl>
  </upgrade>
</database>
```

There are no specific attributes on the `<mssql-db-ddl>` element.

The `<mssql-db-ddl>` element has the following subelements. Each of these elements is optional. There is, at most, a single occurrence of the `<mssql-compression>` and `<mssql-filegroups>` elements on the `<mssql-db-ddl>` element. There can be, however, multiple occurrences of the `<mssql-table-ddl>` element.

`mssql-compression`	Specifies compression settings for SQL Server database tables and indexes at the global, database level. See “The <mssql-compression> database configuration element” on page 255 for more information.
`mssql-filegroups`	Specifies the mapping between SQL Server database filegroups and PolicyCenter logical tablespaces at the global, database level. See “The <mssql-filegroups> database configuration element” on page 255 for more information.
`mssql-table-ddl`	Specifies SQL Server database DDL options for a named table. These settings override values set at the global, database level. See “The <mssql-table-ddl> database configuration element” on page 256 for more information.

See also

- “The database configuration file” on page 237
- “The <upgrade> database configuration element” on page 250
- *Installation Guide*

The <mssql-compression> database configuration element

The `<mssql-db-ddl>` element in file `database-config.xml` contains, at most, a single occurrence of subelement `<mssql-compression>`. The use of the `<mssql-compression>` element is optional. Use this element to set SQL Server database compression options at the global, database level. See also the *Installation Guide*.

The `<mssql-compression>` element has the following syntax.

```
<database>
  <upgrade>
    <mssql-db-ddl>
      <mssql-compression index-compression="NONE|PAGE|ROW" table-compression="NONE|PAGE|ROW/>
    </mssql-db-ddl>
  </upgrade>
</database>
```

The following list describes the attributes that you can configure on the `<mssql-compression>` element.

`index-compression`	If present, specifies the index compression setting for all indexes. Valid values are: • `NONE` • `PAGE` • `ROW` The default is `NONE`.
`table-compression`	If present, specifies the table compression setting for all tables. Valid values are: • `NONE` • `PAGE` • `ROW` The default is `NONE`.

The `<mssql-compression>` element does not contain additional subelements.

See also

- “The database configuration file” on page 237
- “The <upgrade> database configuration element” on page 250

The <mssql-filegroups> database configuration element

The `<mssql-db-ddl>` element in file `database-config.xml` contains, at most, a single occurrence of subelement `<mssql-filegroups>`. The use of the `<mssql-filegroups>` element is optional. Use this attributes to map SQL Server filegroups to PolicyCenter logical tablespaces at the global, database level.

The use of these configuration elements applies to newly created objects only. If you map a specific table to a named filegroup, SQL Server implements the change during table creation or table recreation only. Similarly, if you specify a LOB filegroup for a table or all tables, only newly created tables have their LOB columns stored in the named filegroup. The database does not store newly created LOB columns for existing tables in the named filegroup.

> **Note:** The PolicyCenter schema verification process during server startup reports as warnings any database tables or LOB columns that are not located in a configured filegroup. However, some of the warnings are unavoidable due to the nature of the implementation mechanism and are purely informational.

The `<mssql-filegroups>` element has the following syntax. The following code sample shows required attributes in bold font.

```
<database>
  <upgrade>
    <mssql-db-ddl>
      <mssql-filegroups op="string" admin="string" typelist="string" staging="string"
           index="string" lob="string"/>
    </mssql-db-ddl>
  </upgrade>
</database>
```

The following list describes the attributes that you can configure on the `<mssql-filegroups>` element.

`op`	*Required*. Name of a SQL Server filegroup.
`admin`	*Required*. Name of a SQL Server filegroup.
`typelist`	*Required*. Name of a SQL Server filegroup.
`staging`	*Required*. Name of a SQL Server filegroup.
`index`	*Required*. Name of a SQL Server filegroup.
`lob`	Name of a SQL Server filegroup.

The `<mssql-filegroups>` element does not contain additional subelements.

See also

- "The database configuration file" on page 237
- "The <upgrade> database configuration element" on page 250
- *Installation Guide*

The <mssql-table-ddl> database configuration element

The `<mssql-db-ddl>` element in file `database-config.xml` can contain any number of occurrences of subelement `<mssql-table-ddl>`. The use of the `<mssql-table-ddl>` element is optional. Use this element to specify DDL (Data Definition Language) options for a specific, named SQL Server database table.

The `<mssql-table-ddl>` element has the following syntax. The following code sample shows required attributes in bold font.

```
<database>
  <upgrade>
    <mssql-db-ddl>
    <mssql-table-ddl table-name="string">
      <mssql-index-ddl filter-where="string"index-compression="NONE|PAGE|ROW"
            index-filegroup="string" key-columns="string" partition-scheme="string"/>
      <mssql-table-compression index-compression="NONE|PAGE|ROW"
            table-compression="NONE|PAGE|ROW"/>
      <mssql-table-filegroups lob-filegroups="string" index-filegroup="string"
            table-filegroup="string"/>
      </mssql-table-ddl>
    </mssql-db-ddl>
  </upgrade>
</database>
```

The `<mssql-table-ddl>` element has the following attribute.

`table-name`	*Required*. Name of the table to which these overrides apply.

The `<mssql-table-ddl>` element has the following subelements. Each of these elements is optional. There is, at most, a single occurrence of the `<mssql-table-compression>` and `<mssql-table-filegroups>` elements on the `<mssql-table-ddl>` element. There can be, however, multiple occurrences of the `<mssql-index-ddl>` element.

`mssql-index-ddl`	Specifies DDL options for a specific index. See "The <mssql-index-ddl> database configuration element" on page 257 for more information.
`mssql-table-compression`	Specifies compression for the named table. See "The <mssql-table-compression> database configuration element" on page 257 for more information.
`mssql-table-filegroups`	Specifies a filegroup to associate with a table, index, or LOB. See "The <mssql-table-filegroups> database configuration element" on page 258 for more information.

See also

- "The database configuration file" on page 237
- "The <upgrade> database configuration element" on page 250

The <mssql-index-ddl> database configuration element

The `<mssql-table-ddl>` element in file `database-config.xml` can contain any number of occurrences of subelement `<mssql-index-ddl>`. The use of the `<mssql-index-ddl>` element is optional. Use this element to define SQL Server database DDL options for a specific index, based on the key columns. Any value that you set at this level overrides that same value set at the global, database level. You can create multiple `<mssql-index-ddl>` elements on the parent `<mssql-table-ddl>` element, each of which affects a different index.

The `<mssql-index-ddl>` element has the following syntax. The following code sample shows required attributes in bold font.

```
<database>
  <upgrade>
    <mssql-db-ddl>
      <mssql-table-ddl table-name="string">
        <mssql-index-ddl filter-where="string"index-compression="NONE|PAGE|ROW"
              index-filegroup="string"
              key-columns="string" partition-scheme="string"/>
      </mssql-table-ddl>
    </mssql-db-ddl>
  </upgrade>
</database>
```

The following list describes the attributes that you can configure on the `<mssql-index-ddl>` element.

`key-columns`	*Required.* Comma-delimited list of key columns, in order. Specify `DESC` after the column name for a descending sort order on the column.
The following attributes are all optional.	
`filter-where`	Specifies an index filter to add after the `WHERE` keyword in the SQL Server `CREATE INDEX ... WHERE` statement. The filter that you create must conform to standard SQL Server rules.
`index-compression`	Specifies the compression setting for the specified index. Valid values are: • `NONE` • `PAGE` • `ROW` If not specified, PolicyCenter uses the SQL Server database default.
`index-filegroup`	Name of the filegroup associated with this index. Do not use this attribute if you supply a value for the `partition-scheme` attribute as the two attributes are mutually exclusive.
`partition-scheme`	Name of a partition scheme for this index. Use of this attribute implies the use of PolicyCenter clustering. Do not use this attribute if you supply a value for the `index-filegroups` attribute as the two attributes are mutually exclusive.

The `<mssql-index-ddl>` element does not contain additional subelements.

See also

- *Installation Guide*

The <mssql-table-compression> database configuration element

The `<mssql-table-ddl>` element in file `database-config.xml` contains, at most, a single occurrence of subelement `<mssql-table-compression>`. The use of the `<mssql-table-compression>` element is optional. Any value that you set at this level overrides that same value set at the global, database level. See also the *Installation Guide*.

The `<mssql-table-compression>` element has the following syntax. The following code sample shows required attributes in bold font.

```
<database>
  <upgrade>
    <mssql-db-ddl>
      <mssql-table-ddl table-name="string">
        <mssql-table-compression index-compression="NONE|PAGE|ROW" table-compression="NONE|PAGE|ROW"/>
      </mssql-table-ddl>
    </mssql-db-ddl>
  </upgrade>
</database>
```

The following list describes the attributes that you can configure on the `<mssql-table-compression>` element. All of these attributes are optional.

`index-compression`	Specifies the index compression setting for the specified index. Valid values are: • NONE • PAGE • ROW If not specified, PolicyCenter uses the database default.
`table-compression`	Specifies the table compression setting for the specified table. Valid values are: • NONE • PAGE • ROW If not specified, PolicyCenter uses the database default.

The `<mssql-table-ddl>` element does not contain additional subelements.

The <mssql-table-filegroups> database configuration element

The `<mssql-table-ddl>` element in file `database-config.xml` contains, at most, a single occurrence of subelement `<mssql-table-filegroups>`. The use of the `<mssql-table-groups>` element is optional. Use this element to associate a filegroup with a table, index, or LOB. Any value that you set at this level overrides that same value set at the global, database level.

The `<mssql-table-filegroups>` element has the following syntax. The following code sample shows required attributes in bold font.

```
<database>
  <upgrade>
    <mssql-db-ddl>
      <mssql-table-ddl table-name="string">
        <mssql-table-filegroups table-filegroup="string" index-filegroup="string"
              lob-filegroups="string"/>
      </mssql-table-ddl>
    </mssql-db-ddl>
  </upgrade>
</database>
```

The following list describes the attributes that you can configure on the `<mssql-table-filegroups>` element. All of these attributes are optional. However, if you do not specify at least one of these attributes, there is no need for this element to be present in `database-config.xml`.

`table-filegroup`	Name of the filegroup to associate with this table.
`index-filegroup`	Name of the filegroup to associate with any indexes on this table.
`lob-filegroup`	Name of the filegroup to associate with any large object (LOB, CLOB, or spatial column).

The `<mssql-table-filegroups>` element does not contain additional subelements.

The <ora-db-ddl> database configuration element

The `<upgrade>` element in file `database-config.xml` contains, at most, a single occurrence of subelement `<ora-db-ddl>`. The use of the `<ora-db-ddl>` element is optional. Use this element to set Oracle database DDL (Data Definition Language) options during the creation of new objects in the database. This configuration applies to the database at a global level.

The `<ora-db-ddl>` element has the following syntax. The following code sample shows required attributes in bold font.

```
<database>
  <upgrade
    <ora-db-ddl>

       <!-- Sets Oracle database options at the global, database level -->
      <ora-compression index-compression="true|false" table-compression="ADVANCED|BASIC|NONE"/>
      <ora-lobs caching="true|false" type="BASIC|SECURE|SECURE_COMPRESSED/>
      <tablespaces admin="string" index="string" lob="string" op="string" staging="string"
            typelist="string"/>

       <!-- Sets Oracle options for the named table, overrides values set at the database level -->
      <ora-table-ddl table-name="string">
        <ora-index-ddl index-compression="true|false" index-tablespace="string" key-columns="string"/>
        <ora-lobs caching="true|false" type="BASIC|SECURE|SECURE_COMPRESSED/>>
        <ora-table-compression index-compression="true|false" table-compression="ADVANCED|BASIC|NONE">
        <ora-table-date-interval-partitioning datecolumn="string"
              interval="DAILY|MONTHLY|QUARTERLY|WEEKLY|YEARLY">
        <ora-table-hash-partitioning hash-columns="string" num-partitions="integer"/>
        <ora-table-tablespaces index-tablespace="string" lob-tablespace="string"
              table-tablespace="string"/>
      </ora-table-ddl>

     </ora-db-ddl>
  </upgrade>
</database>
```

There are no specific attributes on the `<ora-db-ddl>` element.

The subelements on the `<ora-db-ddl>` element have the following meanings.

`ora-compression`	Specifies Oracle compression settings for all tables and indexes at the global, database level. See “The <ora-compression> database configuration element” on page 259 for more information.
`ora-lobs`	Specifies attributes for LOB columns on all tables at the global, database level. See “The <ora-lobs> database configuration element” on page 260 for more information.
`ora-table-ddl`	Specifies DDL parameters and overrides for a specific, named Oracle database table. See “The <ora-table-ddl> database configuration element” on page 261 for more information.
`tablespaces`	Specifies default mappings for Oracle tablespaces at a global, database level. See “The <tablespaces> database configuration element” on page 261 for more information.

See also

- “The database configuration file” on page 237
- “The <upgrade> database configuration element” on page 250
- *Installation Guide*

The <ora-compression> database configuration element

The `<ora-db-ddl>` element in file `database-config.xml` contains, at most, a single occurrence of subelement `<ora-compression>`. The use of the `<ora-compression>` element is optional. Use this element to set compression on Oracle database indexes and tables at a global, database level.

The `<ora-compression>` element has the following syntax.

```
<database>
  <upgrade>
    <ora-db-ddl>
      <ora-compression index-compression="true|false" table-compression="ADVANCED|BASIC|NONE"/>
    </ora-db-ddl>
  </upgrade>
</database>
```

The following list describes the attributes that you can configure on the `<ora-compression>` element. All of these attributes are optional.

`index-compression`	*Boolean*. Whether to use index compression for all indexes in an Oracle database. The default is `false`.
`table-compression`	Specifies table compression type for all tables in an Oracle database. Valid values are: • `ADVANCED` • `BASIC` • `NONE` The default is `NONE`.

The `<ora-compression>` element does not contain additional subelements.

See also

- "The database configuration file" on page 237
- "The <upgrade> database configuration element" on page 250
- *Installation Guide*

The <ora-lobs> database configuration element

The `<ora-db-ddl>` element in file `database-config.xml` contains, at most, a single occurrence of subelement `<ora-lobs>`. The use of the `<ora-lobs>` element is optional. Use this element to set options for LOB columns on tables in an Oracle database at a global, database level.

The `<ora-lobs>` element has the following syntax.

```
<database>
  <upgrade>
    <ora-db-ddl>
      <ora-lobs caching="true|false" type="BASIC|SECURE|SECURE_COMPRESSED/>
    </ora-db-ddl>
  </upgrade>
</database>
```

The following list describes the attributes that you can configure on the `<ora-lobs>` element. All of these attributes are optional.

`caching`	*Boolean*. Whether to use caching for all LOB columns on a table or for the Oracle database globally. The default is `false`.
`type`	Sets LOB type globally for the database. Valid values are: • `BASIC` • `SECURE` • `SECURE_COMPRESSED` The default is `SECURE`. **Note:** `SECURE` and `SECURE_COMPRESSED` refer to the use of Oracle SecureFiles LOBs.

The `<ora-lobs>` element does not contain additional subelements.

See also

- "The database configuration file" on page 237
- "The <upgrade> database configuration element" on page 250

The <tablespaces> database configuration element

The `<ora-db-ddl>` element in file `database-config.xml` contains a single occurrence of subelement `<tablespaces>`. You must provide a `<ora-tablespaces>` subelement if using the `<ora-db-ddl>` element. Use this element to map Oracle tablespaces to PolicyCenter logical tablespaces at a global, database level.

The `<tablespaces>` element has the following syntax. The following code sample shows required attributes in bold font.

```
<database>
  <upgrade>
    <ora-db-ddl>
      <tablespaces admin="string" index="string" lob="string" op="string" staging="string"
            typelist="string"/>
    </ora-db-ddl>
  </upgrade>
</database>
```

The following list describes the attributes that you can configure on the `<tablespaces>` element.

The following attributes are all required.	
`admin`	Name of a PolicyCenter logical tablespace.
`index`	Name of a PolicyCenter logical tablespace.
`op`	Name of a PolicyCenter logical tablespace.
`staging`	Name of a PolicyCenter logical tablespace.
`typelist`	Name of a PolicyCenter logical tablespace.
The following attribute is optional.	
`lob`	Name of a PolicyCenter logical tablespace.

The `<tablespaces>` element does not contain additional subelements.

See also

- "The database configuration file" on page 237
- "The <upgrade> database configuration element" on page 250
- *Installation Guide*

The <ora-table-ddl> database configuration element

The `<ora-db-ddl>` element in file `database-config.xml` can contain any number of occurrences of subelement `<ora-table-ddl>`. The use of the `<ora-table-ddl>` element is optional. Use this element to set DDL parameters and overrides for a specific, named table in an Oracle database.

The `<ora-table-ddl>` element has the following syntax. The following code sample shows required attributes in bold font.

```
<database>
  <upgrade>
    <ora-db-ddl>
      <ora-table-ddl table-name="string">
        <ora-index-ddl index-compression="true|false" index-tablespace="string" key-columns="string"/>
        <ora-lobs caching="true|false" type="BASIC|SECURE|SECURE_COMPRESSED/>>
        <ora-table-compression index-compression="true|false" table-compression="ADVANCED|BASIC|NONE">
        <ora-table-date-interval-partitioning datecolumn="string"
              interval="DAILY|MONTHLY|QUARTERLY|WEEKLY|YEARLY">
        <ora-table-hash-partitioning hash-columns="string" num-partitions="integer"/>
        <ora-table-tablespaces index-tablespace="string" lob-tablespace="string"
              table-tablespace="string"/>
      </ora-table-ddl>
    </ora-db-ddl>
  </upgrade>
</database>
```

The `<ora-table-ddl>` element has the following attribute.

`table-name`	*Required*. Name of the table to which these overrides apply.

The subelements on the `<ora-table-ddl>` element have the following meanings.

`ora-index-ddl`	Specifies options for a specific Oracle index, based on key columns. See "The <ora-index-ddl> database configuration element" on page 262 for more information.
`ora-lobs`	Specifies options for LOB columns on a specific, named table in an Oracle database. See "The <ora-lobs> database configuration element" on page 263 for more information.
`ora-table-compression`	Specifies compression options on a specific, named index or table in an Oracle database. See "The <ora-table-compression> database configuration element" on page 264 for more information.
`ora-table-date-interval-partitioning`	Specifies options for date range partitioning on a specific, named table in an Oracle database. See "The <ora-table-date-interval-partitioning> database configuration element" on page 264 for more information.
`ora-table-hash-partitioning`	Specifies options for hash partitioning of a specific, named table in an Oracle database. See "The <ora-table-hash-partitioning> database configuration element" on page 265 for more information.
`ora-table-tablespaces`	Specifies tablespace options for a specific, named table in an Oracle database. See "The <ora-table-tablespaces> database configuration element" on page 265 for more information.

See also

- "The database configuration file" on page 237
- "The <upgrade> database configuration element" on page 250

The <ora-index-ddl> database configuration element

The `<ora-table-ddl>` element in file `database-config.xml` can contain any number of occurrences of subelement `<ora-index-ddl>`. The use of the `<ora-index-ddl>` element is optional. Use this element to set DDL parameters and overrides for a specific Oracle index, based on key columns. See also the *Installation Guide*.

The `<ora-index-ddl>` element has the following syntax. The following code sample shows required attributes in bold font.

```
<database>
  <upgrade>
    <ora-db-ddl>
    <ora-table-ddl table-name="string">
      <ora-index-ddl index-compression="true|false" index-tablespace="string" key-columns="string"/>
    </ora-table-ddl>
    </ora-db-ddl>
  </upgrade>
</database>
```

The following list describes the attributes that you can configure on the `<ora-index-ddl>` element.

`key-columns`	*Required*. Ordered, comma-delimited list of key columns. Specify `DESC` after the column name for a descending sort order on that column.

The following attributes are optional.	
`index-compression`	Specifies the index compression for this index. If you do not specify this attribute, PolicyCenter uses the table or database default.
`index-tablespaces`	Name of the tablespace override for the index.

The subelements on the `<ora-index-ddl>` element have the following meanings.

`ora-index-partitioning`	Defines partitioning for the specified Oracle index. The `<ora-index-partitioning>` element has the following attributes: • `num-hash-partitions` – The number of hash partitions to define. The default is 128. • `partitioning-type` – Required if using this element. Sets the partitioning type to one of the following: ◦ `LOCAL` – Inherit the partitioning type from the table ◦ `HASH` – Use hash partitions. If you set this attribute to `HASH`, then you need to specify the number of partitions to use attribute `num-hash-partitions`. Do not set `partitioning-type` to `HASH` if you specify an `<ora-index-range-partition>` subelement. ◦ `RANGE` – Specify the range partitioning column list and the partition upper limits with one or more `ora-index-range-partition` elements. • `range-partitioning-column-list` – *Optional*. Use to specify the global range partitioning column list. This attribute requires the definition of one or more `ora-index-range-partitioning` elements. Do not specify the last range which is always `VALUES LESS THAN (MAXVALUE)`. Do not use if you set attribute `partitioning-type` to HASH. The `<ora-index-partitioning>` element contains a single subelement: • `ora-index-range-partition` – *Optional*. A comma-delimited, ordered list of literal values corresponding to the column list in the `range-partitioning-column-list` attribute. Use single quotes with string values. PolicyCenter uses this value in the clause `VALUES LESS THAN(`*`value_list`*`)`. Do not use if you set attribute `partitioning-type` to HASH.

See also

- *Installation Guide*

The <ora-lobs> database configuration element

The `<ora-table-ddl>` element in file `database-config.xml` contains, at most, a single occurrence of subelement `<ora-lobs>`. The use of the `<ora-lobs>` element is optional. Use this element to set attributes for LOB columns on a specific, named table in an Oracle database. Any value that you set at this level overrides that same value set at the global, database level.

The `<ora-lobs>` element has the following syntax. The following code sample shows required attributes in bold font.

```
<database>
  <upgrade>
    <ora-db-ddl>
      <ora-table-ddl table-name="string">
        <ora-lobs caching="true|false" type="BASIC|SECURE|SECURE_COMPRESSED/>>
      </ora-table-ddl>
    </ora-db-ddl>
  </upgrade>
</database>
```

The following list describes the attributes that you can configure on the `<ora-lobs>` element. All of these attributes are optional.

`caching`	Sets the LOB cache attribute for the named Oracle table. The default is `false`.

`type`	Sets the LOB type for the named Oracle table. Valid values are: • `BASIC` • `SECURE` • `SECURE_COMPRESSED` The default is `SECURE`. **Note:** `SECURE` and `SECURE_COMPRESSED` refer to the use of Oracle SecureFiles LOBs.

The `<ora-lobs>` element does not contain additional subelements.

The <ora-table-compression> database configuration element

The `<ora-table-ddl>` element in file `database-config.xml` contains, at most, a single occurrence of subelement `<ora-table-compression>`. The use of the `<ora-table-compression>` element is optional. Use this element to set compression on a specific, named index or table in an Oracle database. Any value that you set at this level overrides that same value set at the global, database level. See also the *Installation Guide*.

The `<ora-table-compression>` element has the following syntax. The following code sample shows required attributes in bold font.

```
<database>
  <upgrade>
    <ora-db-ddl>
      <ora-table-ddl table-name="string">
        <ora-table-compression index-compression="true|false" table-compression="ADVANCED|BASIC|NONE">
      </ora-table-ddl>
    </ora-db-ddl>
  </upgrade>
</database>
```

The following list describes the attributes that you can configure on the `<ora-table-compression>` element. All of these attributes are optional.

`index-compression`	*Boolean*. Whether to use index compression. Valid values are: • `true` – Use compression for all indexes on this table. • `false` – Do not use compression for the indexes on this table. If you do not specify this attribute, PolicyCenter uses the database default.
`table-compression`	Specifies table compression for this table. Valid values are: • `ADVANCED` • `BASIC` • `NONE` If you do not specify this attribute, PolicyCenter uses the database default.

The `<ora-table-compression>` element does not contain additional subelements.

The <ora-table-date-interval-partitioning> database configuration element

The `<ora-table-ddl>` element in file `database-config.xml` contains, at most, a single occurrence of subelement `<ora-table-date-interval-partitioning>`. The use of the `<ora-table-date-interval-partitioning>` element is optional. Use to add date range partitioning to a specific, named table in an Oracle database.

The `<ora-table-date-interval-partitioning>` element has the following syntax. The following code sample shows required attributes in bold font.

```
<database>
  <upgrade>
    <ora-db-ddl>
      <ora-table-ddl table-name="string">
        <ora-table-date-interval-partitioning datecolumn="string"
              interval="DAILY|MONTHLY|QUARTERLY|WEEKLY|YEARLY">
      </ora-table-ddl>
    </ora-db-ddl>
  </upgrade>
</database>
```

The following list describes the attributes that you can configure on the `<ora-table-date-interval-partitioning>` element.

`datecolumn`	*Required*. Name of the column to use for the date range. The column must be non-nullable and one of the following types: • `datetime` • `dateonly`
`interval`	*Required*. The interval for each partition. Valid values are: • `DAILY` • `MONTHLY` • `QUARTERLY` • `WEEKLY` • `YEARLY`

The `<ora-table-date-interval-partitioning>` element does not contain additional subelements.

The <ora-table-hash-partitioning> database configuration element

The `<ora-table-ddl>` element in file `database-config.xml` contains, at most, a single occurrence of subelement `<ora-table-hash-partitioning>`. The use of the `<ora-table-hash-partioning>` element is optional. Use this element to add hash partitioning to a table in an Oracle database.

The `<ora-table-hash-partitioning>` element has the following syntax. The following code sample shows required attributes in bold font.

```
<database>
  <upgrade>
    <ora-db-ddl>
      <ora-table-ddl table-name="string">
        <ora-table-hash-partitioning hash-columns="string" num-partitions="integer"/>
      </ora-table-ddl>
    </ora-db-ddl>
  </upgrade>
</database>
```

The following list describes the attributes that you can configure on the `<ora-table-hash-partitioning>` element. All of these attributes are optional.

`hash-column`	Name of the column to use for the hash function: • For keyable entities, the default is the entity ID. • For non-keyable entities, you must provide a value.
`num-partitions`	The number of hash partitions to define. The default is 128.

The `<ora-table-hash-partitioning>` element does not contain additional subelements.

The <ora-table-tablespaces> database configuration element

The `<ora-table-ddl>` element in file `database-config.xml` contains, at most, a single occurrence of subelement `<ora-table-tablespaces>`. The use of the `<ora-table-tablespaces>` element is optional. Use this element to provide overrides for the default table, index, and LOB tablespaces for a specific, named table in an Oracle database.

The `<ora-table-tablespaces>` element has the following syntax. The following code sample shows required attributes in bold font.

```
<database>
  <upgrade>
```

```
    <ora-db-ddl>
      <ora-table-ddl table-name="string">
        <ora-table-tablespaces index-tablespace="string" lob-tablespace="string"
              table-tablespace="string"/>
      </ora-table-ddl>
    </ora-db-ddl>
  </upgrade>
</database>
```

The following list describes the attributes that you can configure on the `<ora-table-tablespaces>` element. All of these attributes are optional.

`index-tablespace`	Name of the tablespace override for the specified index.
`lob-tablespace`	Name of the tablespace override for the specified LOB column.
`table-tablespace`	Name of the tablespace override for the specified table.

The `<ora-table-tablespaces>` element does not contain additional subelements.

The <versiontriggers> database configuration element

The `<upgrade>` element in file `database-config.xml` contains, at most, a single occurrence of subelement `<versiontriggers>`. The use of the `<versiontriggers>` element is optional. Use this element to provide overrides for one or more named database version triggers.

Note: For Oracle, there is an important exception to the ability of the `<versiontriggers>` element to provide overrides for its `<versiontrigger>` subelements. Although the parent element, `<upgrade>`, might have defined the `ora-parallel-dml` and `ora-parallel-query` attributes, these values are not applied to version checks which are included as `<versiontrigger>` subelements. To get a version check to run with parallel query in Oracle, you must list it explicitly, add it as a subelement of the `<versiontriggers>` element, and indicate `parallel-query="true"` on that `<versiontrigger>` subelement.

For example: `<versiontrigger name="com.guidewire.cc.system.database.upgrade.check.RecoveryCategoryNullForPaymentsAndReservesVersionCheck" parallel-query="true"/>`. Other version triggers (but not version checks) that are not listed will inherit the parallel setting defined on the `<upgrade>` element. Note that even though `RecoveryCategoryNullForPaymentsAndReservesVersionCheck` is a version check, it must be included as `<versiontrigger>` element.)

In summary, version checks are exempted from Oracle parallelism unless you explicitly list them. Version triggers are not.

The database upgrade executes a series of version triggers that make changes to the database to upgrade between PolicyCenter versions. Usually, the default settings are sufficient. Change these settings only while investigating a slow database upgrade.

The `<versiontriggers>` element has the following syntax. The following code sample shows required attributes in bold font.

```
<database>
  <upgrade>
    <versiontriggers dbmsperfinfothreshold="integer">
      <versiontrigger extendedquerytracingenabled="true|false" name="string"
            parallel-dml="true|false" parallel-query="true|false"
            queryoptimizertracingenabled="true|false" recordcounters="true|false"
            updatejoinorderedhint="true|false" updatejoinusemergehint="true|false"
            updatejoinusenlhint="true|false"/>
    </versiontriggers>
  </upgrade>
</database>
```

The `<versiontriggers>` element has the following attribute, which is optional.

`dbmsperfinfothreshold`	Specifies–for all version triggers–the threshold after which the database upgrader gathers performance information from the database. The default is 600 (seconds). If a version trigger takes longer than `dbmsperfinfothreshold` number of seconds to execute, PolicyCenter: • Queries the underlying database management system (DBMS). • Builds a set of HTML pages with performance information for the interval in which the version trigger was executing. • Includes these HTML pages in the upgrader instrumentation for the version trigger. You can completely turn off the collection of database snapshot instrumentation for version triggers by setting the value of the `dbmsperfinfothreshold` attribute to 0. If you do not have the license for the Oracle Diagnostics Pack, you must set `dbmsperfinfothreshold` to 0 before running the upgrade.

The `<versiontriggers>` element has the following subelement, of which there can be multiple occurrences.

`versiontrigger`	Provides override instructions for a specific, named, version trigger. See “The <versiontrigger> database configuration element” on page 267

See also

- “The database configuration file” on page 237
- “The <upgrade> database configuration element” on page 250

The <versiontrigger> database configuration element

The `<versiontriggers>` element in file `database-config.xml` can contain any number of occurrences of subelement `<versiontrigger>`. The use of the `<versiontrigger>` element is optional. Use this element to provide specific override instructions for a named version trigger.

The `<versiontriggers>` element has the following syntax. The following code sample shows required attributes in bold font.

```
<database>
  <upgrade>
    <versiontriggers>
      <versiontrigger extendedquerytracingenabled="true|false" name="string"
            parallel-dml="true|false" parallel-query="true|false"
            queryoptimizertracingenabled="true|false" recordcounters="true|false"
            updatejoinorderedhint="true|false" updatejoinusemergehint="true|false"
            updatejoinusenlhint="true|false"/>
    </versiontriggers>
  </upgrade>
</database>
```

The following list describes the attributes that you can configure on the `<versiontrigger>` element.

`name`	*Required.* Case-sensitive, fully qualified name of a version trigger.
The following attributes are all optional.	
`extendedquerytracingenabled`	*Boolean*. (Oracle) Whether PolicyCenter uses extended SQL tracing (Oracle event 10046) for the SQL statements that the version trigger executes. The default is `false`. This output can be very useful if debugging certain types of performance problems. The trace files that PolicyCenter generates exist only on the actual database computer. PolicyCenter does not integrate this information into the upgrade instrumentation.
`parallel-dml`	*Boolean*. (Oracle) Whether Oracle executes DML (Data Manipulation Language) statements in parallel for this particular version trigger during the database upgrade.

	Valid values are: • `true` – Execute DML statement in parallel for inserts and updates for this version trigger, unless the `ora-parallel-dml` attribute on the `<upgrade>` element is set to `disable`. • `false` – Disable parallel execution of DML statements for this version trigger, even if set in code or if the `ora-parallel-dml` attribute on the `<upgrade>` element is set to `enable_all`. If not set, Oracle executes DML statements in parallel, if set in the code or the `ora-parallel-dml` attribute on the `<upgrade>` element is set to `enable_all` (default). See "The <upgrade> database configuration element" on page 250 for more information.
`parallel-query`	*Boolean*. (Oracle) Whether a version trigger provides a hint to the optimizer to use parallel queries while executing SQL queries. This can be useful in improving performance as some version triggers read large amounts of data while running their version check. Valid values are: • `true` – Execute parallel SQL queries unless the `ora-parallel-query` attribute on the `<upgrade>` element is set to `disable`. • `false` – Do not execute parallel SQL queries. The default is `false`. See "The <upgrade> database configuration element" on page 250 for more information.
`queryoptimizertracingenabled`	*Boolean*. (Oracle) Whether PolicyCenter uses query optimizer tracing (Oracle event 10053) for the SQL statements that the version trigger executes. The default is `false`. This output can be very useful if debugging certain types of performance problems. The trace files that PolicyCenter generates exist only on the actual database computer. PolicyCenter does not integrate this information into the upgrade instrumentation.
`recordcounters`	*Boolean*. Whether to record the values of DBMS-specific counters at the beginning and end of each execution of the specified version trigger. Valid values are: • `true` – PolicyCenter retrieves the current state of the counters from the underlying DBMS at the beginning of execution of the version trigger. This behavior is dependent on the value of `dbmsperfinfothreshold` on `<versiontriggers>`. if the execution time of the version trigger exceeds the value of `dbmsperfinfothreshold`, PolicyCenter retrieves the state of the counters at the end of the execution of the version trigger. • `false` – PolicyCenter does not retrieve data. The default is `false`. PolicyCenter writes differences to the DBMS-specific instrumentation screens of the upgrade instrumentation. PolicyCenter only persists these values with the upgrade instrumentation if the execution time of the version trigger exceeds the configured threshold. See also "The <versiontriggers> database configuration element" on page 266.
`updatejoinorderedhint`	*Boolean*. (Oracle) Whether to use the `ORDERED` hint if using a SQL `UPDATE` statement with a join operation. The default is `false`.
`updatejoinusemergehint`	*Boolean*. (Oracle) Whether to use the `USE_MERGE` hint if using a SQL `UPDATE` statement with a join operation. The default is `false`.
`updatejoinusenlhint`	*Boolean*. (Oracle) Whether to use the `USE_NL` hint if using a SQL `UPDATE` statement with a join operation. The default is `false`.

The `<version-trigger>` element does not contain additional subelements.

See also

- "The database configuration file" on page 237
- "The <upgrade> database configuration element" on page 250

chapter 18

Database maintenance

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.

IMPORTANT The versions of third-party products that Guidewire supports for this release are subject to change without notice. For current system and patch level requirements, visit the Guidewire Community and search for knowledge article 1005, *Supported Software Components*.

See also

- *Installation Guide*

About the Upgrade and Versions screen

PolicyCenter includes a Server Tools **Upgrade and Versions** screen that provides detailed information about each database upgrade. The **Upgrade and Versions** screen includes information on the following:

- Version number of upgrade
- Status of upgrade
- Type of upgrade
- Start and time of upgrade
- Status of Deferred Upgrade Tasks batch processing

From this screen, you can drill down into more specific details about the upgrade, or download a report of the upgrade details.

See also

- "The Upgrade and Versions screen" on page 397

Database best practices

Guidewire recommends the following best practices for the PolicyCenter application database.

Database tables

For Oracle databases, keep the Oracle default settings as much as possible. For example, do not set a 4K block size. Consult with Guidewire if you want to change the default Oracle settings.

Do not insert data directly into tables managed by PolicyCenter. This can cause the data distribution tool to fail and cause other problems.

Do not add large numbers of `mediumtext` and `CLOB` columns to a table.

Do not add an index outside of PolicyCenter and not declare it in an extension file.

Database maintenance

If you need to perform any database maintenance tasks, such as applying a patch, shut down all PolicyCenter servers that connect to the database. Restart the servers after the database maintenance is complete.

Run consistency checks on the database, especially after importing data.

Back up the database periodically to support disaster recovery options.

Monitor storage performance. If I/O (input/output) times are slower than 10ms, it indicates that there is most likely an issue.

Monitor tablespace size allocations and disk space to ensure that PolicyCenter does not run out of space.

Update database statistics periodically so that the query optimizer selects an efficient plan for executing application queries.

Database location and synchronization

Physically locate the PolicyCenter application server and the database server in the same timezone and geographic location. Locating the two servers in geographically distant locations increases the time that it takes the application server to query the database. It is possible for the latency between the two servers to impact performance. As a measure of this metric, Guidewire logs the time that it takes to verify access to the database at server start-up.

Synchronize the application server with the database clock. The maximum time difference allowed between the application server and database server is 29 minutes. If you use database clustering, synchronize all nodes in the database cluster with each other.

See also

- “Guidewire database direct update policy” on page 271
- “PolicyCenter database back up” on page 272
- “Database consistency checks” on page 272
- “Understanding database statistics” on page 285

Understanding data model updates

As PolicyCenter starts, 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 PolicyCenter contains a new object extension added after 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.

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 the startup process, PolicyCenter again verifies the data model against the physical database. 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.

Note: If, for any reason, there is an interruption to the server during a database update, 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.

See also

- "Run a schema verification report" on page 271
- "About the Upgrade and Versions screen" on page 269
- "View an upgrade report" on page 399
- "Understanding Guidewire software versioning" on page 400
- *Configuration Guide*

Run a schema verification report

Procedure

1. Ensure that PolicyCenter is running.
2. Open a command prompt and navigate to the following location in the PolicyCenter installation directory:

   ```
   admin/bin
   ```

3. Enter the following command to generate a database schema verification report.

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

   The PolicyCenter administration command prompt tools all require that you enter the password of an administrative user for the tool to work. The use of a user name is optional.

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. 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. Therefore, 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.

Internal application logic, embedded in PolicyCenter application code and APIs, maintains a variety of data and metadata that relates 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 can leave the data in an inconsistent state. Guidewire can require you to restore the database to a previous state if you require support after performing such an update query. Guidewire Support is not able to assist you with diagnosing and correcting application issues caused by your database queries. It is your responsibility to restore PolicyCenter to a consistent state.

WARNING Guidewire supports the in-built automatic database upgrade process only for Guidewire InsuranceSuite products. Guidewire explicitly does not support any alternative process that executes SQL DDL commands on the database.

If you have a legitimate need to update underlying application data, Guidewire recommends that you use Guidewire APIs, either Java or Gosu, to perform the necessary updates. This ensures that you do not miss any critical side effects of the updates 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, before using the APIs, Guidewire strongly recommends that you review your intended updates with your Guidewire Support Partner and/or Guidewire Professional Services team.

In the rare case in which no API exists to correct a data corruption problem, Guidewire can advise you on the SQL queries to use to correct these problems. In these cases, the SQL queries used to update the database must be written, or approved, by Guidewire. This process ensures that all SQL queries use correct logic and that you take all potential side effects into account.

Do not apply any other SQL queries to modify data in a PolicyCenter database. Guidewire does not provide, nor review, such queries for situations in which an API or supported alternate method is available.

PolicyCenter database back up

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 PolicyCenter server in maintenance mode before performing the backup.

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 need to restart PolicyCenter from the last database backup.

After restoring a PolicyCenter database from a backup, instruct PolicyCenter to rebuild its database statistics. See “Understanding database statistics” on page 285.

Keep a backup copy of the PolicyCenter configuration files

Besides 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. Guidewire stores all configuration files in the `PolicyCenter/modules/configuration/config` directory, making it easy to keep these files in a source control system, if desired.

Database consistency checks

PolicyCenter includes a number of database consistency checks that you can run. These checks determine if any unusual conditions exist in the PolicyCenter database such as orphaned child records or inconsistencies between properties. If found, Guidewire reports any consistency check issues in the application console and also in the `pclog.log` file, if configured. These reports are especially useful during trial periods or for testing converted data.

The PolicyCenter log lists a check type for each consistency check that it 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*.

See also

- “The Consistency Checks screen” on page 364

Recommendations for running database consistency checks

Guidewire recommends that you run the database consistency checks on a regular basis to identify potential problems. For example, run database consistency checks:

- Before importing data into a production database
- Before and after a database upgrade
- Weekly after a new PolicyCenter deployment
- Monthly after stabilization of a new PolicyCenter deployment
- Monthly while testing imported data that you converted from a legacy system
- Monthly after you have moved the converted data to a production database

Consistency checks and performance

For very large databases, running consistency checks can have a major negative effect on PolicyCenter performance. For example, some consistency check queries use a large amount of temporary space on Oracle

For development environments with very small data sets, you can enable consistency checks to run each time the PolicyCenter server starts. However, be aware that running consistency tests during server startup:

- Can take a large amount of time to complete
- Can impact performance severely
- Can possibly time out on large datasets

IMPORTANT Set the `checker` attribute on `<database-config>` 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.

Running consistency checks

The following table lists the various ways that you can run or trigger database consistency checks.

Ways to run consistency checks …	For more information
From the **Consistency Checks** screen	“Running consistency checks from PolicyCenter” on page 273
From a command prompt	“Running consistency checks from a command prompt” on page 274
At server start up	“Configure consistency checks to run at server startup” on page 274

See also

- “Configuring the number of threads for consistency checks” on page 274

Running consistency checks from PolicyCenter

Guidewire recommends that you view and run consistency checks from the Server Tools **Consistency Checks** screen in PolicyCenter. The **Consistency Checks** screen lists which consistency checks are available for specific tables. You can also use this page to view the results of previous 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.

See also

- “Run a consistency check from PolicyCenter” on page 367

Running consistency checks from a command prompt

It is possible to launch database consistency checks from a command prompt. Launching consistency checks from the command prompt is useful primarily for scheduling checks to run on a regular basis. This approach to running consistency checks uses the `system_tools -checkdbconsistency` command option.

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.
- Increase the number of worker instance threads used by the consistency check work queue.

See also

- “Configuring the number of threads for consistency checks” on page 274
- “Run a consistency check using system tools” on page 368
- “system_tools command options” on page 429

Configure consistency checks to run at server startup

Procedure

1. Open Guidewire Studio™ for PolicyCenter.
2. In the **Project** window, expand **configuration→config**.
   a. Open file `database-config.xml`.
   b. Add a `checker` attribute on the `<database>` element and set the attribute to `true`.

      ```
      <database checker="true" ...>
      ```

      By default, Guidewire omits this attribute in the base configuration and sets its value to `false`.
   c. Save and close file `database-config.xml`.
3. Restart the application server to trigger the database consistency checks:
   - If working in a development environment, restart the QuickStart server.
   - If working in a production environment, create a new WAR or EAR file and deploy the file to the production server.

See also

- “The database configuration file” on page 237
- “Configure worker threads for consistency checks in config.xml” on page 275

Configuring the 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 you use too many threads, there is a greater chance that current users can experience reduced performance if the database operates with a full load.

It is possible to set the number of threads to use for each consistency check in several different ways, depending on how you configure the consistency checks to run. The following table outlines these differences.

Ways to set thread count	For more information
From **Consistency Checks** screen	"Configure worker threads for consistency checks in work-queue.xml" on page 275
From command prompt	"Configure worker threads for consistency checks in work-queue.xml" on page 275
At server start up	"Configure worker threads for consistency checks in config.xml" on page 275

See also

- "Database consistency checks" on page 272
- "The Consistency Checks screen" on page 364

Configure worker threads for consistency checks in work-queue.xml

About this task

PolicyCenter uses the work queue mechanism to run consistency checks.

Procedure

1. In the PolicyCenter Studio **Project** window, expand **configuration→config→workqueue**:
   a. Open file `work-queue.xml` for editing.
   b. Locate the block of code that contains the following `workQueueClass` value.

   ```
   com.guidewire.pl.system.database.checker.DBConsistencyCheckWorkQueue
   ```

   c. Locate the `<worker>` element for this code block.
   d. Set the `instances` and `batchsize` attributes as appropriate for your PolicyCenter installation.
   For example, the following code sets the consistency check work queue to use five worker threads, with each worker checking out ten work items at a time.

   ```
   <worker instances="5" batchsize="10"/>
   ```

2. Restart the application server for your changes to take effect:
   - If working in a development environment, restart the QuickStart server.
   - If working in a production environment, create a new WAR or EAR file and deploy the file to the production server.

See also

- "Configuring work queues" on page 124

Configure worker threads for consistency checks in config.xml

About this task

If the database connection times out while running consistency checks at server startup, Guidewire recommends that you increase the number of worker threads available for processing the consistency checks.

Procedure

1. Open Guidewire Studio™ for PolicyCenter.
2. In the **Project** window, expand **configuration→config**:
   a. Open file `config.xml` for editing.
   b. Locate the `ConsistencyCheckerThreads` parameter.
   In the base configuration, the value of this parameter is 1.

c. Set the value of this parameter to a number that meets your business needs.

   For example, set this value to five.

```
<param name="ConsistencyCheckerThreads" value="5"/>
```

3. Restart the application server for your changes to take effect:
   - If working in a development environment, restart the QuickStart server.
   - If working in a production environment, create a new WAR or EAR file and deploy the file to the production server.

Resize database columns

About this task

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, the following commonly used procedure provides a general outline of the steps involved making this kind of database change.

IMPORTANT Guidewire does not support re-sizing any database columns that are part of the PolicyCenter base configuration.

Procedure

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, it is possible that you 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.

Purging unwanted data

Over time, the PolicyCenter database acquires and stores ever-increasing amounts of data. Left unchecked, the database can contain a large amount of unused and unnecessary data. Guidewire recommends that you periodically remove this stale data to improve performance and reduce complexity during upgrade. To facilitate the removal of unnecessary data, Guidewire provides a number of purge-related batch processes and work queues.

About purging activity-related workflow data

Each time PolicyCenter creates an activity, it adds that activity to the following tables:

- `pc_Workflow`
- `pc_WorkflowLog`
- `pc_WorkflowWorkItem`

After a user completes the activity, PolicyCenter sets the workflow status to completed. PolicyCenter never uses the `pc_Workflow`, `pc_WorkflowLog` and `pc_WorkflowWorkItem` table entry for that activity again. These tables grow in size over time and can adversely affect performance and waste disk space. Excessive records in these tables also negatively impacts the performance of the database upgrade.

Guidewire recommends that you periodically purge workflows, workflow log entries, and workflow items for completed activities to improve database upgrade and operational performance and to recover disk space.

See also

- “Purging workflow data” on page 283
- “Purging workflow log data” on page 283
- “Purging work item set data” on page 284

Purging batch process history data

Process History Purge batch processing deletes batch process history data from the PolicyCenter `ProcessHistory` table. The process uses configuration parameter `BatchProcessHistoryPurgeDaysOld` to determine the number of days to retain batch process history before deletion. By default, Guidewire sets the value of `BatchProcessHistoryPurgeDaysOld` to 45 days.

In the base configuration, PolicyCenter schedules the `ProcessHistoryPurge` batch process to run on the third day of the month at 3:30 a.m.

It is possible to launch Process History Purge batch processing from within PolicyCenter, or, directly from a command prompt.

PolicyCenter	Navigate to the Server Tools **Batch Process Info** screen and run the **Process History Purge** batch process.
Command prompt	Launch the Purge Workflows batch process from the `PolicyCenter/admin/bin` directory with the following command: `maintenance_tools -password *password* -startprocess ProcessHistoryPurge` The PolicyCenter administration command prompt tools all require that you enter the password of an administrative user for the tool to work. The use of a user name is optional.

See also

- “Process History Purge batch process” on page 148
- “The Batch Process Info screen” on page 353
- “maintenance_tools command” on page 426

Purging cluster member data

Purge Cluster Members batch processing deletes `ClusterMemberData` entities from the PolicyCenter database. The process uses Gosu class `PurgeClusterMembers`, in conjunction with configuration parameter `ClusterMemberPurgeDaysOld`, to determine the date on which to delete a `ClusterMemberData` entity. By default, Guidewire sets the value of `PurgeClusterMembers` to 30days.

In the base configuration, PolicyCenter schedules the `PurgeClusterMember` batch process to run on the first day of each month, at 2:00 a.m.

It is possible to launch Purge Cluster Members batch processing from within PolicyCenter, or, directly from a command prompt.

PolicyCenter	Navigate to the Server Tools **Batch Process Info** screen and run the **Purge Cluster Members** batch process.
Command prompt	Launch the Purge Workflows batch process from the `PolicyCenter/admin/bin` directory with the following command: `maintenance_tools -password *password* -startprocess PurgeClusterMembers` The PolicyCenter administration command prompt tools all require that you enter the password of an administrative user for the tool to work. The use of a user name is optional.

See also

- "Purge Cluster Members batch process" on page 149
- "The Batch Process Info screen" on page 353
- "maintenance_tools command" on page 426

Purging failed work items

Purge Failed Work Items batch processing deletes failed work items from all work queues. The process uses Gosu class `PurgeFailedWorkItems` to determine which work items to delete.

In the base configuration, PolicyCenter schedules the `PurgeFailedWorkItems` batch process to run on the first day of the month, at 1:00 a.m.

It is possible to launch Purge Failed Work Items batch processing from within PolicyCenter, or, directly from a command prompt.

PolicyCenter	Navigate to the Server Tools **Batch Process Info** screen and run the **Purge Failed Work Items** batch process.
Command prompt	Launch the Purge Workflows batch process from the `PolicyCenter/admin/bin` directory with the following command: `maintenance_tools -password password -startprocess PurgeFailedWorkItems` The PolicyCenter administration command prompt tools all require that you enter the password of an administrative user for the tool to work. The use of a user name is optional.

See also

- "Purge Failed Work Items batch process" on page 149
- "The Batch Process Info screen" on page 353
- "maintenance_tools command" on page 426

Purging job and policy period data

Purge batch processing purges jobs and prunes policy periods that meet the purge and prune criteria from the PolicyCenter database. This process deletes jobs and other entities from the PolicyCenter database.

In the base configuration, PolicyCenter schedules the `Purge` work queue to run at 4:30 a.m. every night. However, Guidewire disables Purge batch processing in the base configuration. To use this batch processing type, you must first enable it.

It is possible to launch Purge batch processing from within PolicyCenter, or, directly from a command prompt.

PolicyCenter	Navigate to the Server Tools **Batch Process Info** screen and run the **Purge** batch process.
Command prompt	Launch the Purge work queue from the `PolicyCenter/admin/bin` directory with the following command: `maintenance_tools -password password -startprocess Purge` The PolicyCenter administration command prompt tools all require that you enter the password of an administrative user for the tool to work. The use of a user name is optional.

See also

- "Reset Purge Status and Check Dates work queue" on page 155
- "Purge work queue" on page 149
- "The Batch Process Info screen" on page 353
- "maintenance_tools command" on page 426
- *Configuration Guide*

Purging message history data

Purge Message History batch processing deletes old messages from the PolicyCenter `MessageHistory` table. Configuration parameter `KeepCompletedMessagesForDays` sets the length of time that a message remains in the message history table before the batch process considers a message for deletion from the database. By default, Guidewire sets the value of `KeepCompletedMessagesForDays` to 90 days.

In the base configuration, PolicyCenter schedules the `PurgeMessageHistory` batch process to run on the 20th of the month, at 1:00 a.m.

It is possible to launch Purge Message History batch processing from within PolicyCenter, or, directly from a command prompt.

PolicyCenter	Navigate to the Server Tools **Batch Process Info** screen and run the **Purge Message History** batch process.
Command prompt	Launch the Purge Message History batch process from the `PolicyCenter/admin/bin` directory with the following command: `maintenance_tools -password` *`password`* `-startprocess PurgeMessageHistory` The PolicyCenter administration command prompt tools all require that you enter the password of an administrative user for the tool to work. The use of a user name is optional.

See also

- "Purge Message History batch process" on page 150
- "The Batch Process Info screen" on page 353
- "maintenance_tools command" on page 426

Purging old transaction ID data

Purge Old Transaction IDs batch processing deletes SOAP header transaction IDs generated by systems external to PolicyCenter. Guidewire does not schedule this batch process in the base configuration as the table that stores the transaction IDs takes very little space in the database. Unless there is a constant buildup of these transaction IDs, there is no real need to continually purge this data. In fact, if you do purge this data, it is then not possible to determine if a new transaction is a duplicate of a transaction sent by the external system at an earlier date.

It is possible to launch Purge Old Transaction IDs batch processing from within PolicyCenter, or, directly from a command prompt.

PolicyCenter	Navigate to the Server Tools **Batch Process Info** screen and run the **Purge Old Transaction IDs** batch process.
Command prompt	Launch the Purge Old Transaction IDs batch process from the `PolicyCenter/admin/bin` directory with the following command: `maintenance_tools -password` *`password`* `-startprocess PurgeTransactionIDs` The PolicyCenter administration command prompt tools all require that you enter the password of an administrative user for the tool to work. The use of a user name is optional.

See also

- "The work queue scheduler" on page 120
- "Purge Old Transaction IDs batch process" on page 150
- "The Batch Process Info screen" on page 353
- "maintenance_tools command" on page 426

Purging orphaned policy period data

Purge Orphaned Policy Periods batch processing finds orphaned policy periods (policy periods not associated with a specific job) and deletes them from the PolicyCenter database. The process deletes policy periods and other entities from the PolicyCenter database.

PolicyCenter does not schedules the `PurgeTransactionIDs` work queue in the base configuration. You must either run this process manually or add the work queue to file `scheduler-config.xml`.

It is possible to launch Purge Orphaned Policy Periods batch processing from within PolicyCenter, or, directly from a command prompt.

PolicyCenter	Navigate to the Server Tools **Batch Process Info** screen and run the **Purge Orphaned Policy Periods** batch process.
Command prompt	Launch the Purge Orphaned Policy Periods work queue from the `PolicyCenter/admin/bin` directory with the following command: `maintenance_tools -password *password* -startprocess PurgeOrphanedPolicyPeriods` The PolicyCenter administration command prompt tools all require that you enter the password of an administrative user for the tool to work. The use of a user name is optional.

See also

- "The work queue scheduler" on page 120
- "The Batch Process Info screen" on page 353
- "maintenance_tools command" on page 426
- *Configuration Guide*

Purging profiler data

Purge Profiler Data batch processing deletes profiler data from the PolicyCenter database. This process uses the read-only `ProfilerDataPurgeBatchProcess` class, in conjunction with configuration parameter `ProfilerDataPurgeDaysOld`, to determine how many days to retain profiler data before deleting it. By default, Guidewire sets the value of `ProfilerDataPurgeDaysOld` to 30 days.

PolicyCenter does not schedules the `PurgeProfilerData` batch process in the base configuration. You must either run this process manually or add the batch process to file `scheduler-config.xml`.

It is possible to launch Purge Profiler Data batch processing from within PolicyCenter, or, directly from a command prompt.

PolicyCenter	Navigate to the Server Tools **Batch Process Info** screen and run the **Purge Profiler Data** batch process.
Command prompt	Launch the Purge Profiler Data batch process from the `PolicyCenter/admin/bin` directory with the following command: `maintenance_tools -password *password* -startprocess PurgeProfilerData` The PolicyCenter administration command prompt tools all require that you enter the password of an administrative user for the tool to work. The use of a user name is optional.

See also

- "The work queue scheduler" on page 120
- "Purge Profiler Data batch process" on page 151
- "The Batch Process Info screen" on page 353
- "maintenance_tools command" on page 426

Purging quote clones

Purge Quote Clones batch processing deletes quote clones (copies of policy period quotes) from the PolicyCenter database. This work queue requires that you write your own implementation of the `PolicyPeriodQuoteClonePlugin` plugin interface and register it in the Plugins registry.

PolicyCenter does not schedules the `PurgeQuoteClones` work queue in the base configuration. You must either run this process manually or add the work queue to file `scheduler-config.xml`.

It is possible to launch Purge Quote Clones batch processing from within PolicyCenter, or, directly from a command prompt.

PolicyCenter	Navigate to the Server Tools **Batch Process Info** screen and run the **Purge Quote Clones** batch process.
Command prompt	Launch the Purge Quote Clones work queue from the `PolicyCenter/admin/bin` directory with the following command:

```
maintenance_tools -password password -startprocess PurgeQuoteClones
```

The PolicyCenter administration command prompt tools all require that you enter the password of an administrative user for the tool to work. The use of a user name is optional.

See also

- "The work queue scheduler" on page 120
- "Purge Quote Clones work queue" on page 151
- "The Batch Process Info screen" on page 353
- "maintenance_tools command" on page 426
- *Configuration Guide*

Purging rate book export data

Purge Rate Book Export Result batch processing removes Excel and XML files associated with `RateBookExportResult` objects from the PolicyCenter database. Configuration parameter `RateBookExportResultAgeForPurging` sets the length of time to retain these files before deletion. By default, Guidewire sets the value of `RateBookExportResultAgeForPurging` to 60 days.

In the base configuration, PolicyCenter schedules the `PurgeRateBookExportResult` work queue to run every day at 3:00 a.m.

It is possible to launch Purge Rate Book Export Result batch processing from within PolicyCenter, or, directly from a command prompt.

PolicyCenter	Navigate to the Server Tools **Batch Process Info** screen and run the **Purge Rate Book Export** Result batch process.
Command prompt	Launch the Purge Rate Book Export Result batch process from the `PolicyCenter/admin/bin` directory with the following command: `maintenance_tools -password password -startprocess PurgeRateBookExportResult` The PolicyCenter administration command prompt tools all require that you enter the password of an administrative user for the tool to work. The use of a user name is optional.

See also

- "Purge Rate Book Export Result work queue" on page 152
- "The Batch Process Info screen" on page 353
- "maintenance_tools command" on page 426

Purging rating worksheets data

Purge Rating Worksheets batch processing removes `WorksheetContainer` objects from the PolicyCenter database. Configuration parameter `RatingWorksheetContainerAgeForPurging` sets the minimum number days after the closure of a job before the `WorksheetContainer` associated with its policy is eligible for deletion from the database. By default, Guidewire sets the value of `RatingWorksheetContainerAgeForPurging` to90 days.

PolicyCenter does not schedules the `PurgeWorksheets` work queue in the base configuration. You must either run this process manually or add the work queue to file `scheduler-config.xml`.

It is possible to launch Purge Rating Worksheets batch processing from within PolicyCenter, or, directly from a command prompt.

PolicyCenter	Navigate to the Server Tools **Batch Process Info** screen and run the **Purge Rating Worksheets** batch process.
Command prompt	Launch the Purge Rating Worksheets work queue from the `PolicyCenter/admin/bin` directory with the following command: `maintenance_tools -password password -startprocess PurgeWorksheets` The PolicyCenter administration command prompt tools all require that you enter the password of an administrative user for the tool to work. The use of a user name is optional.

See also

- "Purge Rating Worksheets work queue" on page 152
- "The Batch Process Info screen" on page 353
- "maintenance_tools command" on page 426

Purging temporary risk assessment data

Purge Risk Assessment Temporary Store batch processing deletes temporary objects created for risk assessment from the PolicyCenter database. Configuration parameter `PurgeRiskAssessmentTempStoreDays` specifies the length of time to retain the objects before deletion. By default, Guidewire sets the value of `PurgeRiskAssessmentTempStoreDays` to 30 days.

In the base configuration, PolicyCenter schedules the `PurgeRiskAssessmentTempStore` work queue to run every Monday at 1:00 a.m.

It is possible to launch Purge Risk Assessment Temporary Store batch processing from within PolicyCenter, or, directly from a command prompt.

PolicyCenter	Navigate to the Server Tools **Batch Process Info** screen and run the **Purge Risk Assessment Temporary Store** batch process.
Command prompt	Launch the Purge Risk Assessment Temporary Store batch process from the `PolicyCenter/admin/bin` directory with the following command: `maintenance_tools -password *password* -startprocess PurgeRiskAssessmentTempStore` The PolicyCenter administration command prompt tools all require that you enter the password of an administrative user for the tool to work. The use of a user name is optional.

See also

- "Purge Risk Assessment Temporary Store work queue" on page 152
- "The Batch Process Info screen" on page 353
- "maintenance_tools command" on page 426
- *Configuration Guide*

Purging temporary policy period data

Purge Temporary Policy Periods batch processing removes `PolicyPeriod` objects with a status of Temporary from the PolicyCenter database. Whenever you start a new policy transaction (job) or create a new policy revision, PolicyCenter creates a `PolicyPeriod` object. For a short amount of time during object initialization, the policy period has a status of Temporary. If an error occurs during initialization of the job or policy period, it is possible for the policy period to remain in the PolicyCenter database with a status of Temporary.

Guidewire disables Purge Temporary Policy Periods batch processing in the base configuration. To enable this batch process, set configuration parameter `PurgeTemporaryPolicyPeriodsEnabled` to `true`.

Configuration parameter `PurgeTemporaryPolicyPeriodsAfterDays` specifies the length of time to retain the temporary object before deletion. By default, Guidewire sets the value of `PurgeTemporaryPolicyPeriodsAfterDays` to 14 days.

PolicyCenter does not schedules the `PurgeTemporaryPolicyPeriods` work queue in the base configuration. You must either run this process manually or add the work queue to file `scheduler-config.xml`.

It is possible to launch Purge Temporary Policy Periods work batch processing from within PolicyCenter, or, directly from a command prompt.

PolicyCenter	Navigate to the Server Tools **Batch Process Info** screen and run the **Purge Temporary Policy Periods** batch process.
Command prompt	Launch the Purge Temporary Policy Periods work queue from the `PolicyCenter/admin/bin` directory with the following command: `maintenance_tools -password *password* -startprocess PurgeTemporaryPolicyPeriods`

The PolicyCenter administration command prompt tools all require that you enter the password of an administrative user for the tool to work. The use of a user name is optional.

See also

- "Purge Temporary Policy Periods work queue" on page 153
- "The Batch Process Info screen" on page 353
- "maintenance_tools command" on page 426

Purging workflow data

Purge Workflow batch processing deletes any completed workflows, after resetting any referenced workflows. This process uses Gosu class `PurgeWorkflow`, in conjunction with configuration parameter `WorkflowPurgeDaysOld`, to determine the number of days to retain workflow data before deletion. By default, Guidewire sets the value of `WorkflowPurgeDaysOld` to 60 days, the number of days since the last update to the workflow (the date the workflow completed).

In the base configuration, PolicyCenter schedules the `PurgeWorkflows` batch process to run on the first day of each month, at 1:30 a.m.

It is possible to launch the Purge Workflow batch process from within PolicyCenter, or, directly from a command prompt.

PolicyCenter	Navigate to the Server Tools **Batch Process Info** screen and run the **Purge Workflow** batch process.
Command prompt	Launch the Purge Workflows batch process from the `PolicyCenter/admin/bin` directory with the following command: `maintenance_tools -password `*`password`*` -startprocess PurgeWorkflows` The PolicyCenter administration command prompt tools all require that you enter the password of an administrative user for the tool to work. The use of a user name is optional.

See also

- "Purge Workflow batch process" on page 153
- "The Batch Process Info screen" on page 353
- "maintenance_tools command" on page 426

Purging workflow log data

Purge Workflow Logs batch processing deletes completed workflow logs. It does not remove workflow records, only workflow log records. The process uses Gosu class `PurgeWorkflowLogs`, in conjunction with configuration parameter `WorkflowLogPurgeDaysOld`, to determine the number of days to retain the workflow logs before deletion. By default, Guidewire sets the value of `WorkflowLogPurgeDaysOld` to 30 days.

In the base configuration, PolicyCenter schedules the `PurgeWorkflowLogs` batch process to run on the first day of each month, at 2:30 a.m.

It is possible to launch the Purge Workflow Logs batch process from within PolicyCenter, or, directly from a command prompt.

PolicyCenter	Navigate to the Server Tools **Batch Process Info** screen and run the **Purge Workflow Logs** batch process.
Command prompt	Launch the Purge Workflow Logs batch process from the `PolicyCenter/admin/bin` directory with the following command: `maintenance_tools -password `*`password`*` -startprocess PurgeWorkflowlogs` The PolicyCenter administration command prompt tools all require that you enter the password of an administrative user for the tool to work. The use of a user name is optional.

See also

- "Purge Workflow Logs batch process" on page 153
- "The Batch Process Info screen" on page 353
- "maintenance_tools command" on page 426

Purging work item set data

Work Item Set Purge batch processing deletes work item sets from the database. The process uses Gosu class `WorkItemSetPurge`, in conjunction with configuration parameter `BatchProcessHistoryPurgeDaysOld`, to specify the number of days to retain work item sets before deletion.

In the base configuration, PolicyCenter schedules the `WorkItemSetPurge` batch process to run on the second day of each month, at 1:30 a.m.

It is possible to launch the Work Item Set Purge batch process from within PolicyCenter, or, directly from a command prompt.

PolicyCenter	Navigate to the Server Tools **Batch Process Info** screen and run the **Work Item Set Purge** batch process.
Command prompt	Launch the Purge Workflow Logs batch process from the `PolicyCenter/admin/bin` directory with the following command: `maintenance_tools -password` *`password`* `-startprocess WorkItemSetPurge` The PolicyCenter administration command prompt tools all require that you enter the password of an administrative user for the tool to work. The use of a user name is optional.

See also

- "Work Item Set Purge batch process" on page 157
- "The Batch Process Info screen" on page 353
- "maintenance_tools command" on page 426

Purging work queue instrumentation data

Work Queue Instrumentation Purge batch processing deletes instrumentation data for work queues from the PolicyCenter database. The process uses Gosu class `WorkQueueInstrumentationPurge`, in conjunction with configuration parameter `InstrumentedWorkerInfoPurgeDaysOld`, to determine how long to retain work queue instrumentation before deletion. By default, Guidewire sets the value of `InstrumentedWorkerInfoPurgeDaysOld` to 45 days.

In the base configuration, Guidewire schedules the `WorkQueueInstrumentationPurge` batch process to run on the second day of the month, at 2:30 a.m.

It is possible to launch the Work Queue Instrumentation Purge batch process from within PolicyCenter, or, directly from a command prompt.

PolicyCenter	Navigate to the Server Tools **Batch Process Info** screen and run the **Work Queue Instrumentation Purge** batch process.
Command prompt	Launch the Purge Workflows batch process from the `PolicyCenter/admin/bin` directory with the following command: `maintenance_tools -password` *`password`* `-startprocess WorkQueueInstrumentationPurge` The PolicyCenter administration command prompt tools all require that you enter the password of an administrative user for the tool to work. The use of a user name is optional.

See also

- "Work Queue Instrumentation Purge batch process" on page 158
- "The Batch Process Info screen" on page 353
- "maintenance_tools command" on page 426

chapter 19

Database statistics generation

This topic discusses database statistics, metadata that describe the underlying database.

Understanding database statistics

Database statistics are metadata that describe the underlying database. For example, database statistics store row counts in a table, the distribution of data in the 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.

Use the Server Tools **Database Statistics** screen to determine if any statistics are out of date. Guidewire recommends that you archive database statistics as standard practice. This ensures that you have a record of the database history that you can review, if it becomes necessary.

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

Database statistics generation for Oracle databases

There are several different ways in which to generate database statistics in Guidewire PolicyCenter if using an Oracle database:

- Use the Oracle Autotask infrastructure to manage the task of gathering database table statistics.
- Run PolicyCenter batch processing DBStats periodically to collect database table statics.

You enable the use of each method by setting the value of the `useoraclestatspreferences` attribute on the `<tablestatistics>` element in file `database-config.xml`.

Statistics gathering	`useoraclestatspreferences`	Description	For more information
Oracle AutoTask	`true`	Disable DBStats batch processing and use Oracle AutoTask to manage the collection of database statistics.	"Using Oracle AutoTask for statistics generation" on page 292
`DBStats` batch processing	`false`	(Default) Disable Oracle AutoTask and use DBStats to manage the collection of database statistics.	"Database Statistics work queue" on page 138

Note: A change in the value of `useoraclestatspreferences` takes effect only during an application upgrade.

If using `DBStats` batch processing to manage the collection of database statistics:

- Do not execute Oracle `dbms_stats` manually.
- Manually execute, or schedule, `DBStats` batch processing.
- Disable the automatic generation of database statistics using Oracle by doing one of the following:
  - Disable the Oracle AutoTask "auto optimizer stats collection" automated task.
  - Set the AUTOSTATS_TARGET preference to ORACLE. This action ensures that the automated task gathers statistics for the Oracle Dictionary only.

Guidewire recommendations for Oracle database installations

- Guidewire recommends that Oracle implementations only update database statistics during quiet periods, such as weekends or nights, so that these updates do not occur while PolicyCenter is under heavy load. By default, updating statistics on a table or index invalidates existing query plans related to that table or index.
- Guidewire recommends that Oracle implementations use the `NO_INVALIDATE => AUTO_INVALIDATE` option while updating database 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.

Disable automatic database statistics generation by Oracle

About this task

If you do not intend to use the Oracle AutoTask infrastructure to manage the collection of database table statistics, then disable the generation of database statistics by Oracle before you install Guidewire PolicyCenter. If Oracle database statistics is enabled after the installation of PolicyCenter, do the following:

Procedure

1. Disable Oracle database statistics generation.
2. Delete the schema statistics.
3. Gather full database statistics using the Guidewire `DBStats` batch process.

Database statistics generation for SQL Server databases

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

Guidewire requires that you ensure the following options are set to true on the SQL Server database used for PolicyCenter:

- `Auto Create Statistics`
- `Auto Update Statistics`

Updates to database statistics

Updating database statistics can take a long time on a large database. Collect full statistics only if there are significant changes to data such as after a major upgrade. Guidewire recommends that you collect full statistics if using the `zone_import` command or if there are performance problems. Under standard operating conditions, you generally need to gather incremental database statistics only on a frequent basis.

If you encounter performance problems or degradation related to the database, check the **Database Statistics** screen on the **Info Pages** section of the **Server Tools**. If this screen shows suspicious or inaccurate statistics, update database statistics. If the data change is high, consider using a daily schedule for updating incremental statistics. Use the daily schedule to provide statistics on tables that have had a configurable percentage of data changed since the last statistics process was run.

Understanding database statistics batch processing

PolicyCenter uses Database Statistics batch processing `DBStats` to generate database statistics. The database statistics batch process is resource-intensive. To prevent an accidental start of this process, you cannot start Database Statistics from the Server Tools **Batch Process Info** screen. Instead, you must start the process manually from a command prompt.

IMPORTANT Consult with your Database Administrator before starting the database statistics process.

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.

- “Database Statistics work queue” on page 138
- “The Batch Process Info screen” on page 353
- “maintenance_tools command” on page 426

Automatic generation of database statistics during upgrade

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 full database statistics during the next maintenance window after performing a major upgrade.

Managing database statistics using system tools

You can use the `system_tools` command options to explicitly update database statistics or to generate the SQL statements to update statistics. It is possible to update database statics fully or incrementally.

Full database statistics	Generates database statistics for every table in the PolicyCenter database.
Incremental database statistics	Generates database statistics for tables for which the change in the table data caused by inserts and deletes exceeds a certain percentage threshold. You specify this threshold through the `incrementalupdatethresholdpercent` attribute on the `<databasestatistics>` element in file `database-config.xml`. The default is 10 percent.

It is possible to pause the database statistics updating process, just as you can with other work queues. Use the Server Tools **Work Queue Info** page to pause an in-progress work queue.

See also

- “Understanding database statistics” on page 285
- “system_tools command” on page 429

Perform a full database statistics update

Procedure

1. Ensure that the PolicyCenter is running.
2. In a command prompt, navigate to the following location in the PolicyCenter installation directory:

   ```
   admin/bin
   ```

3. Enter the following command to update statistics for all tables.

   ```
   system_tools -password password -updatestatistics description false
   ```

   The PolicyCenter administration command prompt tools all require that you enter the password of an administrative user for the tool to work. The use of a user name is optional.

Update statistics on tables that exceed a change threshold

About this task

This process does not update statistics on any table that contains locked statistics.

Procedure

1. Ensure that the PolicyCenter is running.
2. Open a command prompt and navigate to the following location in the PolicyCenter installation directory:

   ```
   admin/bin
   ```

3. Enter the following command to update statistics for tables exceeding the change threshold.

   ```
   system_tools -password password -updatestatistics description true
   ```

   The PolicyCenter administration command prompt tools all require that you enter the password of an administrative user for the tool to work. The use of a user name is optional.

Check the database statistics updating process

Procedure

1. Ensure that the PolicyCenter is running.
2. Open a command prompt and navigate to the following location in the PolicyCenter installation directory:

   ```
   admin/bin
   ```

3. Enter the following command to check on the state of the process that updates database statistics.

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

   The PolicyCenter administration command prompt tools all require that you enter the password of an administrative user for the tool to work. The use of a user name is optional.

Cancel the database statistics updating process

About this task

If you receive the following error on submitting a new database statistics job, it is possible that the Database Statistics process terminated abruptly:

```
The Database Statistics process is already in progress
```

If you receive this error message, you need to manually cancel the process.

Procedure

1. Ensure that the PolicyCenter application server is running.
2. Open a command prompt and navigate to the following location in the PolicyCenter installation directory:

   ```
   admin/bin
   ```

3. Enter the following command to cancel the process that updates database statistics.

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

   The PolicyCenter administration command prompt tools all require that you enter the password of an administrative user for the tool to work. The use of a user name is optional.

Generate statistics SQL for all tables

About this task

You can use the results purely as a reference, or you can edit the statements and execute them outside of PolicyCenter.

Procedure

1. Ensure that the PolicyCenter server is running.
2. Open a command prompt and navigate to the following location in the PolicyCenter installation directory:

   ```
   admin/bin
   ```

3. Enter the following command to generate database statistic SQL statements for all tables.

   ```
   system_tools -password password -getdbstatisticsstatements
   ```

   The PolicyCenter administration command prompt tools all require that you enter the password of an administrative user for the tool to work. The use of a user name is optional.

Result

PolicyCenter groups the output statements by table.

Generate statistics SQL for tables that exceed a threshold

Procedure

1. Ensure that the PolicyCenter server is running.
2. Open a command prompt and navigate to the following location in the PolicyCenter installation directory:

   ```
   admin/bin
   ```

3. Enter the following command to generate database statistic SQL statements for tables exceeding the change threshold.

   ```
   system_tools -password password -getincrementaldbstatisticsstatements
   ```

   The PolicyCenter administration command prompt tools all require that you enter the password of an administrative user for the tool to work. The use of a user name is optional.

Result

PolicyCenter groups the output statements by table.

Configuring database statistics generation

You control which database statistics statements PolicyCenter generates by configuring the database connection in the `database-config.xml` file. You control the number of threads that PolicyCenter uses for the work queue used in generating database statistics through configuration of file `work-queue.xml`.

See also

- “Understanding database statistics” on page 285
- “Managing database statistics using system tools” on page 287

Configuring the number of threads for statistics generation

Note: This topic is relevant only if using `DBStats` batch processing to manage the collection of database statistics.

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 configuration, the statistics generation work queue uses five worker threads and each worker checks out ten work items at a time.

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

See also

- “The work queue scheduler” on page 120

Statistics and the database configuration file

File `database-config.xml` contains a single `<databasestatistics>` element, which has the following format.

```
<databasestatistics samplingpercentage="integer" databasedegree="integer"
 incrementalupdatethresholdpercent="integer">
    <tablestatistics name="string" samplingpercentage="integer" databasedegree="integer"
     action="delete|keep|update">
      <histogramstatistics name="string" numbuckets="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="0" databasedegree="4">
    <histogramstatistics name="scheduledsenddate" numbuckets="254"/>
  </tablestatistics>
  <tablestatistics name="pc_table2" action="delete" />
</databasestatistics>
```

The previous example configures the following database statistic generation behavior:

- Collects statistics on all PolicyCenter tables using the automatic sampling size and with a degree of parallelism of 4.
- Samples table `pc_table1` using Oracle `AUTO_SAMPLE_SIZE`, which is the recommended value.
- Uses a parallel degree of 4.
- Defines a histogram with 254 buckets (time slots) on `pc_check.scheduledsenddate`. Guidewire requires that you provide a value for this attribute.
- Deletes statistics on `pc_table2` due to the attribute `action="delete"`.

See also

- "The <databasestatistics> database configuration element" on page 291
- "The <tablestatistics> database configuration element" on page 294

The <databasestatistics> database configuration element

You use the `<databasestatistics>` element in `database-config.xml` to specify 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.
`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. The default is 10.
`numappserverthreads`	On both Oracle and SQL Server, the `numappserverthreads` attribute controls the number of threads that PolicyCenter uses to update database statistics for staging tables during import only. Command prompt tool `table_import` launches this import. The value defaults to 1. If the value is greater than 1, then the PolicyCenter 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. The default is 1.
`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 an integer from `1` to `100` to directly set the `estimate_percent` value. However, Guidewire recommends highly that you set the `samplingpercentage` value to `0` to set `estimate_percent` to `AUTO_SAMPLE_SIZE`. The default value is `0`. 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`. The default is 0.
`useoraclestatspreferences`	On Oracle, this attribute sets the database statistics preferences to be able to use the Oracle Autotask infrastructure instead of the `DBStats` batch process from PolicyCenter. The default is `false`, which requires that you disable the Autotask and schedule `DBStats` batch processing in its place. Changes to the value of this attribute only take effect during an application upgrade.

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.

See also

- "Configuring database statistics generation" on page 290
- "The <databasestatistics> database configuration element" on page 291
- "Using Oracle AutoTask for statistics generation" on page 292
- "table_import command" on page 435

Using Oracle AutoTask for statistics generation

The purpose of the `useoraclestatspreferences` attribute on the `<databasestatistics>` element in file `database-config.xml` is twofold:

- To set the table statistics preferences according to those set for Oracle in file `database-config`.
- To have the Oracle AutoTask infrastructure manage the task of collecting table statistics, based on preferences that Guidewire sets in the base configuration.

To set Oracle to collect table statistics, set this attribute value to `true`, for example:

```
<database name="ExampleDatabase" dbtype="oracle" env="oracle" printcommands="true">
  <databasestatistics databasedegree="4" useoraclestatspreferences="true">
  ...
```

Any change to this attribute value takes effect only during an upgrade, either a full upgrade or a rolling (configuration) upgrade. To force PolicyCenter to recognize the change without an application upgrade, increment the application metadata version and restart the application server.

After you set this attribute to `true`, the next application upgrade does the following:

- It clears all existing preferences for table statistics.
- It resets the preferences for table statistics to those currently defined for Oracle table statistics in `database-config.xml`.
- It creates a new tab named **Oracle Statistics Preferences** in the Server Tools **Info Pages→Database Catalog Statistics Information** screen.

To confirm that the application upgrade set the database statistics preferences, review the **Oracle Statistics Preferences** tab and verify the preferences.

Enabling Oracle AutoTask

after you set this parameter to `true` for the first time, do the following:

- Ensure that the statistics target setting has the default value of `AUTO`.
- Ensure that you enable Oracle AutoTask for statistics collection.

The Oracle DBA can run the following SQL commands to set these value back to their default.

```
EXEC dbms_stats.set_global_prefs('AUTOSTATS_TARGET','AUTO');
EXEC dbms_auto_task_admin.enable(client_name => 'auto optimizer stats collection', operation => NULL,
      window_name => NULL);
```

Gathering new statistics

If you enabled the database before you set `useoraclestatspreferences` to `true`, you need to delete the existing schema statistics and gather new statics. The Oracle DBA can use the following SQL commands for this task.

```
EXEC dbms_stats.delete_schema_stats('PCUSER');
EXEC DBMS_STATS.GATHER_SCHEMA_STATS(ownname=>'PCUSER', options=>'GATHER');
```

Note: Replace *`PCUSER`* with the actual PolicyCenter database user.

After the application upgrade

After you perform an application upgrade with `useoraclestatspreferences` set to `true`, there is no longer any need to run `DBStats` batch processing. At this point, Oracle AutoTask automatically handles statistics collection during the maintenance windows defined for the database. To disable `DBStats` batch processing, remove its schedule from `scheduler-config.xml`.

Future application upgrades

For future application upgrades, Guidewire recommends that you consider setting the `updatestatistics` attribute on the `<upgrade>` element in `database-config.xml` to `true`, for example:

```
<upgrade degree-parallel-ddl="6" degree-of-parallelism="6" ora-parallel-dml="enable_all"
      updatestatistics="true">
```

Setting the `updatestatistics` attribute to `true` allows the PolicyCenter upgrader to create any additional histograms required by the new application version.

For more information, refer to the following Oracle documentation:

- *Oracle Database Administrator's Guide*, "`Managing Automated Database Maintenance Tasks`"
- *Oracle Database SQL Tuning Guide*, "`Managing Optimizer Statistics: Basic Topics`"

Resetting useoraclestatspreferences

Any change to the `useoraclestatspreferences` attribute in `database-config.xml` (from `false` to `true` or from `true` to `false`) takes effect only after an upgrade, either a full upgrade or a rolling (configuration) upgrade. However, if you reset this attribute from `true` to `false`, PolicyCenter throws an exception during the next upgrade and prevents the upgrade from continuing due to locked table statistics in the Oracle database. Review the details of the exception provided in the server log to determine which table statistics need to be unlocked. See "Revert to DBStats batch processing for database statistics" on page 293 for information on how to unlock the table statistics.

See also

- "The Oracle Statistics Preferences tab" on page 377

Revert to DBStats batch processing for database statistics

You must perform a sequence of manual steps if you want to revert to using the table statistics preferences set in `database-config.xml` rather than using Oracle AutoTask infrastructure to manage the task of collecting table statistics. Any change to the `useoraclestatspreferences` attribute in `database-config.xml` (from `false` to `true` or from `true` to `false`) takes effect only after an upgrade, which can be either a full database upgrade or a rolling configuration update. However, if you attempt to reset this attribute from `true` to `false`, PolicyCenter throws an exception during the next upgrade and prevents the upgrade from continuing due to locked statistics in certain Oracle database tables. Review the exception log detail to determine which tables statistics need to be unlocked.

Procedure

1. Reset attribute `useoraclestatspreferences` to `false` in file `database-config`.
2. Start the application server in upgrade mode.

   The upgrade fails due to locked table statistics in the Oracle database.
3. Review the server log for which table statistics need to be unlocked.

   Search for text that is similar to the following:

   ```
   com.guidewire.pl.system.exception.UpgradeException: The following tables have locked statistics...
   ```
4. Unlock the specified table statistics.

   The following example code illustrates what SQL commands the BillingCenter database DBA needs to execute.

   ```
   SET serverout on timing on
   DECLARE
   CURSOR locked_tables_cur IS SELECT table_name
   FROM user_tab_statistics
   ```

```
WHERE STATTYPE_LOCKED IS NOT NULL;
str VARCHAR2(256);
BEGIN
FOR locked_tables IN locked_tables_cur
LOOP
str:= 'EXEC DBMS_STATS.UNLOCK_TABLE_STATS(USER, ''' || locked_tables.table_name|| ''')' ;
dbms_output.put_line(str);
EXECUTE IMMEDIATE str;
END LOOP;
END;
/
```

This step is mandatory. Otherwise, the PolicyCenter upgrade fails.

5. Delete the current table statistics preferences. The following example code illustrates what SQL commands the BillingCenter database DBA needs to execute.

```
EXEC DBMS_STATS.DELETE_SCHEMA_PREFS('PCUSER', 'NO_INVALIDATE');
EXEC DBMS_STATS.DELETE_SCHEMA_PREFS('PCUSER', 'CASCADE');
EXEC DBMS_STATS.DELETE_SCHEMA_PREFS('PCUSER', 'STALE_PERCENT');
EXEC DBMS_STATS.DELETE_SCHEMA_PREFS('PCUSER', 'ESTIMATE_PERCENT');
EXEC DBMS_STATS.DELETE_SCHEMA_PREFS('PCUSER', 'DEGREE');
EXEC DBMS_STATS.DELETE_SCHEMA_PREFS('PCUSER', 'METHOD_OPT');
```

Note: Replace *PCUSER* with the actual PolicyCenter database user.

6. Restart the application server in upgrade mode.
7. After the application upgrade completes, schedule the execution of `DBStats` batch processing to collect database table statistics.

The <tablestatistics> database configuration element

The `<databasestatistics>` element in `database-config.xml` has one subelement, `<tablestatistics>`. You 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` attributes, and statistic gathering behavior of PolicyCenter. You must 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|force"/>
```

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 it 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:

`delete`	Delete the statistics on the table. This value does nothing in SQL Server.
`force`	Update statistics for this table while running incremental statistics, regardless of the value of attribute `incrementalupdatethreshold` on the `<databasestatistics>` element.
`keep`	Keep the existing statistics. PolicyCenter does not update statistics for any table for which the user explicitly specifies `keep` as the value for the `action` attribute. This value affects any type of database.
`update`	Update the statistics for the table: • Full statistics update - Always update • Incremental statistics update - Update only if the change in table data caused by inserts and deletes exceeds the value of attribute `incrementalupdatethreshold` on the `databasestatistics>` element.

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, you must also supply a value for the `action` attribute.

See also

- "Configuring database statistics generation" on page 290
- "The <databasestatistics> database configuration element" on page 291
- "The <histogramstatistics> database configuration element" on page 295

The <histogramstatistics> database configuration element

The `<tablestatistics>` element in `database-config.xml` has several subelements, one of which is the `<histogramstatistics>` element. 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`. (Guidewire sets this value internally.)

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

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

The `name` attribute specifies a column name. The `numbuckets` attribute controls the maximum number of buckets for the specified histogram. Guidewire requires that you provide a value for this attribute. The default value for the number of buckets is 254.

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.

See also

- "Configuring database statistics generation" on page 290
- "The <databasestatistics> database configuration element" on page 291
- "The <tablestatistics> database configuration element" on page 294

chapter 20

Importing and exporting administrative data

Guidewire categorizes certain types of application data as administration data. (Guidewire frequently shortens this tem to just *admin data*.) For example, activity patterns are administration data. This topic provides information related to importing administrative data into, and exporting data from, Guidewire PolicyCenter. While users enter much of the information into PolicyCenter directly, at times, it is more convenient or necessary to enter information in bulk.

Ways to import administrative data

It is possible to import administrative data into Guidewire PolicyCenter by using the following mechanisms.

Mecha-nism	How	Description
PolicyCenter	**Administration** tab	Import and export administrative data files in XML format by using functionality available from the PolicyCenter**Administration** tab. The import functionality provides warnings on collisions between incoming data and existing data and provides a mechanism for you to resolve the collisions. For more information, see: • "Importing and exporting administrative data from PolicyCenter" on page 309
File import	`import_tools` `ImportToolsAPI`	Use the `import_tools` command prompt tool to import one or more CSV or XML files containing administrative data. Use this command to import administrative data only. Guidewire does not support using the `import_tools` command to import other types of data. It is also possible to use the `ImportToolsAPI` web service to import one or more XML files containing administrative data. For more information, see: • "import_tools command" on page 424 • "Using tools to import administrative data" on page 307 • *Integration Guide*
File load	`import` directory	Load data from CSV files contained in the following directory: **configuration→config→import→gen** At initial database upgrade, starting from an empty database, PolicyCenter loads the ad-ministrative data contained in the files in this directory. These files include the default activity patterns, non-renewal question templates, and roles and role privileges.

Mechanism	How	Description
		You can add additional files to this directory for load at initial server startup. For details, see "About the import directory" on page 298. **IMPORTANT** PolicyCenter loads files from the **import→gen** directory only if the database is empty, during the database upgrade at initial server start.
File load	`Security` directory	Load security zone data from file `security-zones.xml`, contained in the following directory: **configuration→config→Security** For information on the syntax to use with this XML-formatted file, see "Importing security zone data" on page 311. **IMPORTANT** PolicyCenter loads data from file `security-zones.xml` only if the database is empty, during the database upgrade at initial server start.
Staging tables	`table_import`	Use the `table_import` command prompt tool to import administrative data from staging tables into PolicyCenter database tables. For details, see "table_import command" on page 435. PolicyCenter supports bulk data import from staging tables for loading zone data only.

IMPORTANT Never modify PolicyCenter operational database tables directly with SQL commands.

About the import directory

After the initial PolicyCenter installation, the database is empty of data. The first time that you start the PolicyCenter application server after installation, PolicyCenter upgrades the database and populates it with data. As part of this initial upgrade, PolicyCenter automatically loads the following administrative data:

- Root group of the user and group tree
- Initial security zone
- Default activity patterns
- Default roles and role permissions (privileges)

PolicyCenter can also import default business rules on initial server startup, depending on the value of configuration parameter `BizRulesImportBootstrapRules.`

Importing administration data at initial server start

During the initial start of the application server with an empty database, PolicyCenter uses the data defined in the following files to load the default activity patterns, roles, and role permissions:

- `activity-patterns.csv`
- `roleprivileges.csv`
- `roles.csv`

These files exist in the following location in Guidewire Studio:

configuration→config →import →gen

It is possible to modify but not remove these files. However, at a bare minimum, these files must define the `superuser` role and its privileges in the base configuration. Otherwise, it is not possible to start PolicyCenter the first time.

Security zone data import at initial server startup

During the initial server startup with an empty database, PolicyCenter loads the security zone data defined in file `security-zones.xml`. The use of this file is optional. You cannot use this mechanism to load security zone data after the initial server startup to populate the database with data.

File `security-zones.xml` exists in the following directory:

configuration→config →Security

See also

- "Importing security zone data" on page 311

Business rules import at initial server startup

During the initial server startup with an empty database, it is possible for PolicyCenter to automatically load business rules defined in the following location in PolicyCenter Studio:

configuration→config →import →bizrules

Configuration parameter `BizRulesImportBootstrapRules` controls this behavior:

- If the value of `BizRulesImportBootstrapRules` parameter is `true`, PolicyCenter imports the business rules defined in this location.
- If the value of `BizRulesImportBootstrapRules` is `false`, PolicyCenter ignores any business rules in that location.

IMPORTANT Guidewire recommends that you set the value of this configuration parameter to `true` in a non-production test environment only.

See also

- "Automatic import of business rules at server startup" on page 346
- *Configuration Guide*

Using file importfiles.txt to import additional admin data

During server startup, PolicyCenter examines file `importfiles.txt` to determine if it lists additional administrative data files to load. This file exists in the following location in PolicyCenter Studio:

configuration→config →import →gen

In the base configuration, file `importfiles.txt` contains the name of one file to load, that of `nonRenewalExplanationPatterns.csv`.

The file format of each file to load must be either CSV or XML. If adding multiple file names to `importfiles.txt`:

- List the files in order of data dependencies.
- Ensure that you provide both the file name and the file extension for each file that you add.

WARNING Any file that you name in file `importfiles.txt` must exist in the **gen** folder or the database upgrade fails.

Import additional admin data at server start

Add additional admin files in CSV or XML format to file `importfiles.txt` to load the file contents into the database at server start.

Procedure

1. Place the appropriately formatted file containing the data in the following folder in Guidewire Studio.

   configuration→config →import →gen

2. Add the files to load to the list of files in `importfiles.txt`.

   If adding multiple file names to `importfiles.txt`:

   - List the files in order of data dependencies.
   - Ensure that you provide both the file name and the file extension for each file that you add.

Result

PolicyCenter loads the data in sequential order of the files listed in file `importfiles.txt` during the next application server start.

About adding admin data after initial server startup

In general, it is not possible to import additional administrative data contained in the files in the **import→gen** directory after the initial database upgrade. PolicyCenter loads the administrative data contained in the **gen** folder only if starting from an empty (non-upgraded) database. Thereafter, PolicyCenter ignores the files in the **gen** directory.

There is one exception. It is possible to add the data in file `roleprivileges.csv` in the **gen** folder using an `import_tools` command option. This command option adds the privileges associated with each PolicyCenter role in the `roleprivileges.csv` file to those that already exist in the database.

See also

- “Add additional roles and privileges after initial server startup” on page 300
- “import_tools command” on page 424

Add additional roles and privileges after initial server startup

Procedure

1. Ensure that the PolicyCenter server is running.
2. Open a command prompt.
3. Navigate to the following location in the PolicyCenter installation directory:

```
admin/bin
```

4. Run the following command to import the data in file `roleprivileges.csv` in the **import→gen** folder:

```
import_tools -password password -privileges
```

The PolicyCenter administration command prompt tools all require that you enter the password of an administrative user for the tool to work. The use of a user name is optional.

See also

- “Character set encoding for file import” on page 300
- “import_tools command” on page 424

Character set encoding for file import

In the base configuration, if you do not supply a value for the `import_tools -charset` option, PolicyCenter assumes a character set encoding of UTF-8 for the import file. It is possible that the file that you want to import contains characters not recognized by the import tools default character encoding.

If this is the case, then you need to set the character set encoding to a more appropriate value. For example, for a file that contains single-byte data as accented characters or characters with umlauts, a `-charset` value of `ISO-8859-1` is possibly more appropriate.

See also

- “import_tools command” on page 424

Maintain data integrity during administrative data import

About this task

Some PolicyCenter administrative data has dependencies on business roles. For example, PolicyCenter associates roles with groups. Therefore, if you export administrative data from one system into another, you must also export all PolicyCenter roles from the old system and import the roles into the new system.

Procedure

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

Administrative data and the PolicyCenter data model

To import or export data from PolicyCenter, you must understand its data model. In particular, you must 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*, accessible in the following location in the PolicyCenter installation directory:

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

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 following command:

```
gwb genDataDictionary
```

See also

- *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.

Typically, a company imports data from multiple external sources. If you do import data from multiple sources, use a naming convention to generate public IDs for external sources. For example, if you import from two systems (`Adminsystem` and `Salessystem`), each source can have a contact entity with ID=5432. Thus, Guidewire recommends that you use the following ID format so 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`. Thus, there is no risk of duplicate IDs. There is also the benefit of knowing from which system the record originated.

Public IDs need to be unique only within objects of the same type. For example, all policy objects must have a different public ID. However, an account and a policy with the same public ID do not conflict with each other. 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 ever want to transfer data from one database to another with the export and import utilities, the two databases must have different public ID prefixes. Set the public ID prefix 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. Thus, the public ID format becomes the following:

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

If you have multiple developers, you must coordinate the public ID prefix so that no public ID prefixes overlap. For example, each engineer can 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.

Constructing a CSV file for import

You can only import data for an entity type that already exists in the Guidewire data model. Each CSV-formatted 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
- Data information.

PolicyCenter treats any line that contains the string `entityid` as a heading.

PolicyCenter considers as data any line:

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

In the example shown, 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.

IMPORTANT Guidewire supports using the `import_tools` command to import administrative data only.

Constructing a heading line in CSV-formatted files

Every CSV-formatted file that you import into PolicyCenter must have a heading line. The 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`
- `entityid`

All 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 following fields follow the three required fields (`type`, `data-set`, and `entityid`):

- `addresstype` – Represents a typelist
- `addressline1` – 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:

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

Constructing data lines in CSV-formatted files

Every CSV-formatted file that you import into PolicyCenter must have a heading line, followed by one or more data lines. The `import_tool` identifies data lines by scanning the file for lines with the following characteristics:

- The line does not contain the three required heading fields.
- The line contains comma-delimited values.
- The line contains a third field that is non-empty.

Each data line represents a single instance of a data model entity.

Data line - field 1

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`.

Data line - field 2

The second field in a data line is the value of the highest-numbered data-set of which the imported object is part.

```
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,,,,,,,,,,,
```

PolicyCenter orders data-sets 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, PolicyCenter requests data-set 10240. PolicyCenter assumes that data-set 10240 includes every data-set that it is possible to create.

You can leave the second field blank, in which case PolicyCenter always includes this object in the import regardless of the requested data-set.

Data line - field 3

The third field in any data line must be the public ID for that particular data object. 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,
       referred,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 does not require the foreign key, PolicyCenter imports the data, sets the foreign key field to null, and reports an error. If PolicyCenter does require the foreign key, 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 that are referenced 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 follows:

```
[ [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]]]
```

Construct an XML file for import

About this task

To create a properly formatted XML file of administrative data for import, Guidewire recommends that you perform the following sequence of steps:

Procedure

1. First, export the current administrative data as an XML file by using the functionality available on the **Export Data** screen available to PolicyCenter administrators. This screen provides the ability to export various types of administrative data in XML format.
2. Modify the file to suit your business needs, carefully preserving the XML formatting for administrative data.
3. Regenerate the XSD files by using the following command in the PolicyCenter installation directory:

   ```
   gwb genImportAdminDataXsd
   ```

It is important to regenerate the XSD files every time that you modify the data model.

4. Re-import the modified file by using either the `import_tools` command or the **Export Data** screen available to PolicyCenter administrators. This screen provides the ability to import administrative data in XML format into PolicyCenter.

See also

- "Importing and exporting administrative data from PolicyCenter" on page 309
- "Constructing the XML for the administrative data import file" on page 306
- "import_tools command" on page 424

Constructing the XML for the administrative data import file

The `PolicyCenter/build/xsd/pc_import.xsd` file defines the XML schema used for import and export of administrative data. This file further references information in two other XSD files in the same directory:

- `pc_entities.xsd`
- `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 by entering the following command in a command prompt open in the PolicyCenter installation folder:

```
gwb genImportAdminDataXsd
```

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

The following XML example shows the default activity pattern **30 Day Diary** from the PolicyCenter administrative export file.

```
<?xml version="1.0"?>
<import xmlns="http://guidewire.com/pc/exim/import" version="p5.86.a12.309.46"
        usePeriodicFlushes="true">
  <ActivityPattern public-id="sample_pattern:19">
    <ActivityClass>task</ActivityClass>
    <AutomatedOnly>false</AutomatedOnly>
    <Category>reminder</Category>
    <Code>30_day_diary</Code>
    <Command/>
    <Description/>
    <Description_L10N_ARRAY/>
    <DocumentTemplate/>
    <EmailTemplate/>
    <EscBusCalLocPath/>
    <EscalationBusCalTag/>
    <EscalationDays/>
    <EscalationHours/>
    <EscalationInclDays/>
    <EscalationStartPt/>
    <Mandatory>false</Mandatory>
    <PatternLevel>All</PatternLevel>
    <Priority>normal</Priority>
    <Recurring>true</Recurring>
    <ShortSubject/>
    <ShortSubject_L10N_ARRAY/>
    <Subject>30 day diary</Subject>
    <Subject_L10N_ARRAY/>
    <TargetBusCalLocPath/>
    <TargetBusCalTag/>
    <TargetDays>30</TargetDays>
    <TargetHours/>
    <TargetIncludeDays>elapsed</TargetIncludeDays>
    <TargetStartPoint>activitycreation</TargetStartPoint>
    <Type>general</Type>
  </ActivityPattern>
</import>
```

You can:
- Modify any existing entry in the administrative export file and re-import the file.
- Add additional entries by using the existing entries as a model and re-import the file.

Foreign key references

Within an XML-formatted administrative data import file, it is common to have references between objects in the file. For example, a user object can reference the group of which it is a member. As the group definition is elsewhere in the XML file, or perhaps the definition exists 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 can 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`.

It is possible to reference an item within a single XML file before you define the item. For example, you can refer to supervisor of a user before you define the supervisor in the file. This enables you to define all groups first.

PolicyCenter reports errors only if a referenced object still does not exist after reading the entire file.

Validating the import XSD file

It is important to regenerate the PolicyCenter XSD files after you modify the data model. These files are likely to change as you configure the data model. You generate these XSD files by opening a command prompt in the PolicyCenter installation directory and entering the following command:

```
gwb genImportAdminDataXsd
```

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

```
uses java.io.File
uses java.io.FileInputStream
uses javax.xml.validation.SchemaFactory
uses javax.xml.XMLConstants
uses javax.xml.parsers.SAXParserFactory
uses org.xml.sax.helpers.DefaultHandler
uses gw.testharness.TestBase

// Provide the correct directory of the pc_import.xsd
var schemaFile = new File("pc_import.xsd")
TestBase.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();
var fis = new FileInputStream("myImportFile.xml")
parser.parse(fis, new DefaultHandler())
```

Using tools to import administrative data

Guidewire provides the `import_tools` command for importing new or updated administrative data into existing tables in the PolicyCenter database. PolicyCenter reads the import data from a comma-separated values (CSV) file or an XML file.

IMPORTANT PolicyCenter supports this command for importing administrative data, but not for importing other types of data.

PolicyCenter uses the public ID of each object in the data import to determine if an object with that public ID already exists in the database. See "Administrative data and the PolicyCenter data model" on page 301 for a discussion of public IDs

During import, if PolicyCenter finds a match in entity public IDs, it does the following:

- If there is no difference between the import record and the database record, PolicyCenter ignores the import record.
- If there are differences between the two records, PolicyCenter overwrites any existing database record values with the values from the import file. PolicyCenter does not throw concurrent data change exception if the imported records overwrite existing records in the database.
- If there are null entries for a record in the import file, PolicyCenter nulls out those values in the record in the database.

IMPORTANT Guidewire supports using the `import_tools` command to import administrative data only.

See also

- "import_tools command" on page 424

Import administrative data using import tools

Before you begin

The `MaximumFileUploadSize` parameter in file `config.xml` must be larger than the size of any file that you attempt to import. The `MaximumFileUploadSize` parameter value is in megabytes (MB). In the base configuration, the default value of `MaximumFileUploadSize` is 20 MB.

Procedure

1. Create a CSV or XML file describing the data, by using one of the following methods:
   - Create the XML or CSV file manually.
   - Export the current administrative data as an XML file from PolicyCenter and modify the file.
2. Import the CSV or XML file by using the `import_tools` command:

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

   The PolicyCenter administration command prompt tools all require that you enter the password of an administrative user for the tool to work. The use of a user name is optional.

   The `-import` option requires that you provide the name of the file to import (*`fileName`*). There are a number of other options that you can set as well.

See also

- "Constructing a CSV file for import" on page 302
- "Construct an XML file for import" on page 305
- "import_tools command" on page 424

Importing and exporting administrative data from PolicyCenter

You can import and export administrative data directly from PolicyCenter by using functionality that is available to PolicyCenter administrators. It is only possible to import and export XML-formatted administrative data through the PolicyCenter interface.

About importing PolicyCenter administrative data

You can import administrative data directly from within PolicyCenter by using the **Import Data** screen available to PolicyCenter administrators. To access this screen, navigate to the following location in PolicyCenter:

Administration→Utilities→Import Data

You can import XML-formatted administrative data only.

During the import, PolicyCenter looks for existing data objects that have public IDs that match those of the data objects being imported. PolicyCenter notifies you if it determines there are public ID matches. You can then chose to do one of the following:

- Overwrite all existing data with the imported data
- Discard updates to any existing data and keep the existing data
- Interactively resolve each data conflict on a case-by-case basis

Importing arrays

PolicyCenter handles arrays differently depending on whether it is importing an owned array or a virtual array:

- **Owned array** – If an entity owns the array, PolicyCenter notifies you of the 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 imported array.
- **Virtual array** – It is not possible replace the contents of a virtual array with those of an imported array as it is not possible to delete a virtual array.

Import administrative data into PolicyCenter

Procedure

1. Log into Guidewire PolicyCenter as a user with the `viewadmin` and `soapadmin` permissions.
2. Select the **Administration** tab.
3. In the left-hand navigation pane, expand **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.

Next steps

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.

About exporting PolicyCenter administrative data

You can export administrative data directly from within PolicyCenter by using the **Export Data** screen available to PolicyCenter administrators. It is possible to export XML-formatted administrative data only.

To access the **Export Data** screen, navigate to the following location in Guidewire PolicyCenter:

Administration→Utilities→Export Data

It is possible to export the following data sets independently:
- Admin
- Roles
- Policy Form Patterns
- Lookup Table Definitions
- Policy Holds
- Underwriter Issue Types

IMPORTANT If a particular data set is not on the export list, you cannot export the data using this function.

Exporting user data

PolicyCenter handles the export of user data differently between development and production systems.

Server mode	User export data
Development mode	The export file contains usernames and the associated passwords in hash code format.
Production mode	The export process strips both username and password information from the export file.

Notes
1. During export or import of users and groups, PolicyCenter also exports or imports any entities referenced by a `User` or `UserRole` object through a foreign key or array.
2. It is not possible to import a previously exported data file from a production environment back into PolicyCenter as it does not contain the required username/password data.

Export administrative data from PolicyCenter

Procedure

1. Log into Guidewire PolicyCenter as a user with the `viewadmin` and `soapadmin` permissions.
2. Select the **Administration** tab.
3. In the left-hand navigation pane, expand **Utilities→Export Data**.
4. Select the data set to export.
5. Click **Export** to download the XML file.

Understanding roles and permissions

A *permission* (or privilege) is a granular task or ability to see or do something within PolicyCenter. A *role* is a named collection of permissions, and, typically, maps to a job function or job title.

PolicyCenter stores role information in file `roles.csv` and permission information in file `roleprivileges.csv`. Within Guidewire Studio **Project** window, these two files exist in the following location:

configuration→config→import→gen

PolicyCenter loads the contents of these two files into the database upon initial database upgrade, at first server startup after installation. See "About the import directory" on page 298 for details on how PolicyCenter works with the files in the **gen** directory.

See also

- "About adding admin data after initial server startup" on page 300

Role definitions

File `roles.csv` contains a list of PolicyCenter roles, along with a human-readable name and description for each role. Within this file, set the `name` and `description` fields to whatever is useful in uniquely identifying the role.

PolicyCenter reads the file, starting with the first row that contains the `entityid` identifier and imports the data into the database.

The following code samples are examples of role definition entries:

```
Roles,
type,data-set,entityid,description,name,carrierinternalrole,roletype
Role,0,superuser,${AdminData.Role.Description.Superuser},${AdminData.Role.Name.Superuser},true,user
Role,0,underwriter_supervisor,${AdminData.Role.Description.Underwriter_Supervisor},$
{AdminData.Role.Name.Underwriter_Supervisor},true,user
Role,0,underwriter,${AdminData.Role.Description.Underwriter},${AdminData.Role.Name.Underwriter},true,user
...
```

Notice the use of display keys inside the variable construction `${...}` to set the role name and description. Guidewire recommends this approach as it permits the easy localization of these values.

Role permission definitions

File `roleprivileges.csv` contains the mappings that link roles to a set of permissions. PolicyCenter reads the file starting with the first row that contains the `entityid` identifier and imports the data into the database.

The following code samples are examples of permission definition entries:

```
type,data-set,entityid,permission,role
RolePrivilege,0,sample_data:2,abcreate,k
RolePrivilege,0,sample_data:3,abdelete,audit_examiner
RolePrivilege,0,sample_data:4,abedit,audit_examiner
RolePrivilege,0,sample_data:5,abview,audit_examiner
RolePrivilege,0,sample_data:6,anytagcreate,audit_examiner
,,,,
```

Each row in file `roleprivileges.csv` maps a single permission to a role. Each role has multiple permissions and thus multiple rows. For example, the `abcreate` entry grants permission to create a contact to the `audit_examiner` role.

The *PolicyCenter Security Dictionary* provides a full list of role permission, along with a brief description of each. It also provides a list of the correspondences between roles and permissions.

Importing security zone data

In the base default configuration, Guidewire provides a single security zone named Default Security Zone. PolicyCenter imports this default security zone as part of the administration data import into an empty database at initial server startup.

It is possible to add additional security zones to PolicyCenter in the following ways.

File `security-zones.xml`	Define additional security zones in file `security-zones.xml`. PolicyCenter loads this data as part of the administration data load into an empty database at initial server startup. See "Understanding the security zones file" on page 311 for more information.
Export/import of data	Import new security zone data using the export and import functionality accessible in the PolicyCenter**Administration** tab. See "Import security zones" on page 312 for more information.

Understanding the security zones file

Besides importing the default security zone at initial server startup, PolicyCenter also loads any additional security zone data defined in file `security-zones.xml`. The use of this file is optional. You cannot use this mechanism to load security zone data after the initial server startup populates the database with data.

You access file `security-config.xml` in Guidewire Studio™ by navigating in the **Project** window to **configuration→config→Security**.

In the base configuration, file `security-zones.xml` provides only the top-level `<import>` XML element. To be useful, you must add one or more `<SecurityZone>` subelements. There is no limit to the number of `<SecurityZone>` elements that can exist in `security-zone.xml`.

The `<SecurityZone>` element has the following syntax.

```
<SecurityZone public-id="… ">
  <Description>Some meaningful description…</Description>
  <Name>Some meaningful name…</Name>
</SecurityZone>
```

The attributes and subelements of `<SecurityZone>` have the following meanings.

Element	Attribute	Required	Description
`SecurityZone`	`public-id`	Yes	Internal identifier for the security zone. Guidewire recommends that you make the value of this attribute meaningful for your organization. See "Public ID prefix" on page 301 for more information on public IDs.
Name		Yes	External-facing name for the security zone. This name is visible in the PolicyCenter user interface.
`Description`		No	Description of the security zone, which, if provided, is externally facing in the PolicyCenter user interface.

See also

- For information on PolicyCenter import of administration data at initial server startup, see "About the import directory" on page 298.

View the XSD for security zones

About this task

There is no separate XSD for `security-zones.xml`. Instead, view the relevant XSD information in the generated `pc_entities.xsd` file.

Procedure

1. Open a command prompt and navigate to the PolicyCenter installation directory
2. Run the following command:

```
gwb genImportAdminDataXsd
```

3. In the file system, navigate to the following location in the PolicyCenter installation directory:

```
build/xsd
```

4. Open file `pc_entities.xsd`.
5. Search for `SecurityZoneXSDType`.

Import security zones

Procedure

1. Within Guidewire PolicyCenter, navigate to the following location:

   Administration→Utilities→Export Data
2. Select **Admin** from the **Data to Export** drop-down list, then click **Export**.
3. Open the resulting file (`admin.xml`) for editing.
4. Review the file for existing security zones (`<SecurityZone public-id="...">`).

   Pay particular attention to the `public-id` value for each existing security zone. Any security zone that you add to this file must have a unique `public-id` value. See "Public ID prefix" on page 301 for more information on public IDs.

5. Add unique entries for each security zone to import.
   For example:

   ```
   <SecurityZone public-id="pc:236">
     <Description>Some meaningful description…</Description>
     <Name>Some meaningful name…</Name>
   </SecurityZone>
   ```

   This example sets the `public-id` value to `pc:236`. Chose a public ID value that makes business sense for your organization.
6. After saving the file, navigate to the following location in PolicyCenter:
   Administration→Utilities→Import Data
7. Browse to find your modified file and click **Next**.
   If there are conflicts between the administrative data in the import file and the existing administrative data, PolicyCenter provides a mechanism for conflict resolution. You can chose to do one of the following:
   - Overwrite all existing data with the imported data
   - Discard updates to any existing data and keep the existing data
   - Interactively resolve each data conflict on a case-by-case basis
8. After resolving all data conflicts, click **Finish**.

Result

You can now use the updated set of security zones in PolicyCenter without server restart.

See also

- "Construct an XML file for import" on page 305
- "About exporting PolicyCenter administrative data" on page 309
- "About importing PolicyCenter administrative data " on page 309

About importing zone data

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 **Project** window in Guidewire Studio™.

configuration→config→geodata

In the **geodata** folder, PolicyCenter stores the zone information in country-specific `zone-config.xml` files, with each file in its own specific 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

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. The provided zone data files are not complete and might 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.

Importing a zone data file

To import zone data, use the `zone_import` command. The `zone_import` command imports data in CSV format from specified files into database staging tables for zone data. It is only possible to import zone data for a single country at a time. The zone data files that you import must contain zone data for a single country only. To load zone data for multiple countries, use the command multiple times with different, country-specific zone data files each time.

After you use the `zone_import` command to import zone data, set the PolicyCenter server run level to `MAINTENANCE`. Then, move the data from the staging tables to the production tables by using the `table_import` command. See "table_import command" on page 435 for more information on the `table_import` command. Finally, set the PolicyCenter server run level back to `MULTIUSER`.

Guidewire expects that you import address zone data upon first installing PolicyCenter, and then at infrequent intervals thereafter as you receive data updates.

Import a zone data file

Procedure

1. Start the PolicyCenter server.
2. 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 want to load:

   ```
   zone_import -password password -clearstaging [-country countrycode]
   ```

   The PolicyCenter administration command prompt tools all require that you enter the password of an administrative user for the tool to work. The use of a user name is optional.
3. Load the zone data file into the staging tables:

   ```
   zone_import -password password -country countrycode -import filename
   ```
4. Clear existing zone data in production. Perform this step if zone data already exists for the country whose data you intend to load.

   ```
   zone_import -password password [-country countrycode] -clearproduction
   ```
5. Set the PolicyCenter server run level to `MAINTENANCE`:

   ```
   system_tools -password password -maintenance
   ```
6. Load zone data from the staging tables into production:

   ```
   table_import -server url -password password -integritycheckandload -zonedataonly
   ```
7. Set the server run level back to `MULTIUSER`:

   ```
   system_tools -password password -multiuser
   ```

See also

- For information on using the `zone_import` command, see "zone_import command" on page 439.
- For information on using the `table_import` command, see "table_import command" on page 435.
- For information on importing zone data and database staging tables generally, see the *Integration Guide*.
- For information on the web service `ZoneImportAPI` that also imports zone data, see the *Integration Guide*.

Importing custom zone data files

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. PolicyCenter stores the `zone-config.xml` files for base-configuration countries in country-specific folders. Navigate to the following location in the Studio **Project** window to view these files.

configuration→config→geodata

For example, Guidewire provides zone configuration data for France in file `zone-config.xml` in the following location:

configuration→config→geodata→FR

Each country-specific `zone-config.xml` file must have a single top-level `<Zones>` element for that country. Underneath each `<Zones>` element are `<Zone>` elements that define the zone fields to import from zone data files for that 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 the *Globalization Guide*.

chapter 21

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 on which 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.

See also

- *Installation Guide*
- *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 Internal Tools **Free Text Search** screen to execute the batch load command instead. If you run the batch load command with embedded operation, PolicyCenter limits the number of index items to 10,000.

See also

- "The Free-text Search screen" on page 419

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.

Guidewire recommends that you take a backup of the current index before running each free-text batch load command. This backup provides a snapshot of the index that you can restore if the batch load does not complete successfully. For information about how to perform a backup, see the Solr documentation on the following web site:

```
https://archive.apache.org/dist/lucene/solr/ref-guide/apache-solr-ref-guide-4.10.pdf
```

Run the free-text batch load command

About this task

You run the free-text batch load command on the host on which the Guidewire Solr Extension resides. Run the command only if you configured free-text search for external operation.

Procedure

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. Navigate to the following location in the Solr installation:

   ```
   /opt/gwsolr/pc/solr/policy_active/conf
   ```
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 Solr batch load 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 file `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 `workDir` 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

- *Configuration Guide*

chapter 22

Production data fix tool

This topic describes a tightly constrained system for updating data on a running production server other than through PCF pages or web services. Guidewire calls this mechanism the Production Data Fix tool.

WARNING Only use the Production Data Fix tool under extraordinary conditions, with great caution, and upon advice of Guidewire 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 the data change on a production server.

Overview of the production data fix tool

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. Guidewire calls this mechanism the Production Data Fix Tool.

WARNING Only use the Production Data Fix tool under extraordinary conditions, with great caution, and upon advice of Guidewire 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 the data change on a production server.

Separation of permissions

To decrease security risks, the Production Data Fix tool separates its action into separate tasks, each of which has different permissions and entry points.

Permission	Code	Description
Execute a data change	`admindatachangeexec`	Permission to execute the data change code.

Permission	Code	Description
Register a data change	`wsdatachangeedit`	Permission to register a data change Gosu program. Use one of the following methods to register data change code: • The `data_change` command prompt tool • The `DataChangeAPI` WS-I web service
View a data change	`admindatachangeview`	Permission to view the **Data Change** screen.

By having multiple different paths and multiple 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 either the `wsdatachangeedit` permission (to register data change code) or the `admindatachangeexecd` permission (to execute the code), but not both permissions.

Preserving results

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

Replay prevention

To prevent replay attacks, the Production Data Fix tool 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.

The Data Change screen

Administrative users with the `admindatachangeview` permission can view a special PolicyCenter **Data Change** administration screen that displays information about data change operations. To access this page, navigate to the following location in Guidewire PolicyCenter:

Administration→Utilities→Data Change

See also

- "About writing Gosu data change code" on page 323
- "About registering data change code" on page 323
- "Run Gosu data change code" on page 325

Typical use of the production data fix tool

There are several steps in using the Production Data Fix tool. The following list describes these steps.

Task	For more information
Writing and testing the Gosu database code	"About writing Gosu data change code" on page 323
Registering the Gosu database code, either through the use of a system tool or through web services	"About registering data change code" on page 323
Running the Gosu database code	"Run Gosu data change code" on page 325

See also

- "Logging data change operations" on page 325

About writing Gosu data change code

The first stop in using the Production Data Fix tool is to write Gosu code that does the following:

- Correctly and safely makes only necessary changes to the production data
- 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`. 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.

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`.

See also

- *Gosu Reference Guide*

About registering data change code

There are two ways to register your data change code

- Through the `data_change` command prompt tool
- Through the `DataChangeAPI` web service

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.
- A human-readable description for your data change.
- A unique reference ID that you create to represent this data change.

IMPORTANT If you need to re-run a successful data change, you must first re-register the script with a new reference ID. This is requirement preserves the integrity of the results log.

See also

- “About writing Gosu data change code” on page 323

Register a data change from a command prompt

About this task

The command prompt tool `data_change` works by calling the `DataChangeAPI` web service on a running PolicyCenter server.

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

Procedure

1. Ensure that the PolicyCenter application server is running.
2. Open a command prompt and navigate to the following location in the PolicyCenter installation:

   ```
   admin/bin
   ```
3. Run the following command:

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

The PolicyCenter administration command prompt tools all require that you enter the password of an administrative user for the tool to work. The use of a user name is optional.

For example:

```
data_change -description "Fix Employee Name"
      -edit REFID_1234 -gosu c:\PolicyCenter\datachange\gosudatachange_REFID1234.gsp
      -server http://TESTINGSERVER:8180/pc -user su -password gw
```

Result

The script outputs results similar to the following text:

```
Running data_change.gsp
Connecting as su to URL http://localhost:8180/pc/ws/gw/webservice/systemTools/DataChangeAPI
Edit change ref=REFID1234 publicId=cc:1
```

See also

- "Data change command tool reference" on page 326

Register a data change using a web service

About this task

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

Procedure

1. Call the `DataChangeAPI` web service method `updateDataChangeGosu`.
2. Pass the method the following arguments as `String` objects:
   - The reference ID
   - A human-readable description
   - The Gosu code to run

   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)
```

Result

The method call returns the public ID of the new `DataChange` entity instance.

See also

- "Register a data change from a command prompt" on page 323
- "Data change command tool reference" on page 326

Run Gosu data change code

Before you begin

Before executing data change code, confirm that someone created and registered a data change. You must know the reference ID for the registered data change.

About this task

WARNING Only use the Production Data Fix tool under extraordinary conditions, with great caution, and upon advice of Guidewire Support. Before registering a data change on a production server, register and run the data change on a development server first. Guidewire recommends multiple people review and test the code and the results before attempting the data change on a production server.

Procedure

1. Log into your PolicyCenter development environment as an administrator with the `admindatachangeview` permission.
2. Select the **Administration** tab.
3. Expand **Utilities→Data Change**.
4. In the list of data changes, use the **Reference** column to find the data change request by its reference ID.
5. Click on the data change row in that list.

   The screen shows the Gosu code for that data change.
6. Review the Gosu code to confirm it is what you expect.
7. Click **Execute**.

   During code execution, PolicyCenter does not display the results immediately in the **Result** pane. Instead, the status of **Executing** shows in the list of data changes.
8. After a few seconds, click **Reload** on the screen to view the current status and results.
   - If the change is successful, PolicyCenter confirms the success in a message that uses your reference ID:
   - If the change was not successful, PolicyCenter shows any compile errors or exceptions in the user interface in the **Result** pane.
9. Confirm your changes in the database and check your logging results from the change.
10. If the data change appears safe in your development environment, carefully register and run the data change on the production server.

See also

- “About writing Gosu data change code” on page 323

Logging data change operations

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.

Logging entity field-level changes

To log entity field-level 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.

Logging arbitrary text data

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.

Example logging code

The following sample code uses the `setDetailResultWriting` method and the `ResultsWriter` property to create log messages.

```
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 create your writable bundle as the super user.

Data change command tool reference

PolicyCenter provides a tightly constrained system for updating data on a running production server. Guidewire calls this mechanism the Production Data Fix tool. The Production Data Fix tool uses either the `data_change` command option, or, the `DataChangeAPI` web service to make changes to the production data.

Because the `data_change` command allows arbitrary execution of data, Guidewire strongly recommends that you carefully control the ability to create and run code on a production server. The user who runs this command must have permission `wsdatachangeedit`.

WARNING Only use the Production Data Fix tool under extraordinary conditions, with great caution, and upon advice of Guidewire 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 the data change on a production server.

```
data_change -help
data_change -password password [-server url] [-user user] {
            -edit refID -gosu filepath [-description desc] |
            -discard refID |
            -status refID |
            -result refID }
```

See also

- For a description of how and when to use the `data_change` command to change data on a running production server, see "Production data fix tool" on page 321.
- For a description of how to use the `DataChangeAPI` web service, see "Data change web service reference" on page 327.
- For specifics of the `data_change` command options, see "data_change command options" on page 423.

Data change web service reference

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

`DataChangeAPI` method	Description
`updateDataChangeGosu`	Use to register a data change. Pass the method the following arguments as String objects: • The reference ID • A human-readable description • The Gosu code to run The method returns the public ID of the new `DataChange` entity instance. Note the following: • If the data change succeeds with no compile errors, you cannot edit the data change Gosu code. You must re-register the script with a new reference ID to make changes to the code. • If the data change was never run, or had compile errors, you can update (edit) and rerun Gosu code with the same reference ID.
`discardDataChange`	Use to discard a data change that you already registered. Pass the method a data change reference ID as a `String` object. You cannot discard a data change that was already run.
`getDataChangeResult`	Use to review the result of a data change that you already registered. Pass the method a data change reference ID. The method returns a `String` object that represents the results in the `DataChange` entity instance. If running the data change code generated parse errors, the results include the errors.
`getDataChangeStatus`	Use to view the status of a data change that you already registered. Pass the method a data change reference ID. The method returns a `DataChangeStatus` typecode. Valid values include: • `Open` • `Discarded` • `Executing` • `Failed` • `Completed`

See also

- "Overview of the production data fix tool" on page 321
- "Typical use of the production data fix tool" on page 322
- "Data change command tool reference" on page 326
- *Integration Guide*

part 6

Business rules administration

chapter 23

Administering business rules

This topic provides information relevant to administering PolicyCenter business rules.

See also

- “Business rules import and export” on page 339

Business rules in Guidewire PolicyCenter

PolicyCenter provides a management tool that business analysts and rule administrators use to do the following:

- Add, modify, and delete underwriting business rules
- Manage the process of testing and deploying business rules
- Manage the import and export of business rules between development, test, and production PolicyCenter servers

You access and manage business rules for underwriting within Guidewire PolicyCenter in the following location:

Administration→Business Settings → Business Rules→Underwriting Rules

PolicyCenter business rules, managed from and within PolicyCenter, are distinct from Gosu rules, managed through PolicyCenter Studio.

See also

- *Application Guide*
- *Rules Guide*

PolicyCenter business rules roles and permissions

PolicyCenter business rule roles and permissions:

- Govern access to the ClaimCenter application screens that relate to business rules
- Define the permitted functionality available on the business rule screens

A *permission* provides the ability to perform some task such as accessing an application screen or editing the contents of that screen. A *role* is a grouping of permissions that you can assign to a user.

Roles

In the base configuration, PolicyCenter provides the following user role for use with business rules.

Underwriting Rules Viewer	View the **Business Rules→Import/Export Status** screen, from which it is possible to view information or download rules only
Underwriting Rules Editor	View and edit underwriting rules
Underwriting Rules Admin	All permissions related to underwriting business rules

Permissions

In the base configuration, PolicyCenter provides the following permissions for use with underwriting business rules.

Permission	Code	Provides ability to...
View Rules	`uwruleview`	View the **Business Rules→Import/Export Status** screen for the underwriting rules
Edit Rules	`uwruleedit`	Perform the following business rule edit operations: • Create, clone, edit, delete • Revert a rule to a specific rule version • Enable or disable a business rule • Promote a business rule to Staged status
Approve Rules	`uwruleapprove`	Promote a business rule to Approved status.
Deploy Rules	`uwruledeploy`	Deploy approved business rules.
Import Rules	`uwruleimport`	Import business rules into PolicyCenter.

File security-config.xml

Guidewire defines the default permissions for each business rule subtype in file `security-config.xml`. Thus, you see in the base configuration file, the following permissions associated with the Underwriting business rules.

```
<BizRulesPermissions>
  <RuleSubtypeAccessProfile entity="UWRule">
    <RuleViewPermission permission="uwruleview"/>
    <RuleEditPermission permission="uwruleedit"/>
    <RuleDeployPermission permission="uwruledeploy"/>
    <RuleImportPermission permission="uwruleimport"/>
    <RuleApprovePermission permission="uwruleapprove"/>
  </RuleSubtypeAccessProfile>
</BizRulesPermissions>
```

Business rule configuration parameters

PolicyCenter provides the following parameters in file `config.xml` to manage the deployment and configuration of business rules:

- `BizRulesDeploymentEnabled`
- `BizRulesDeploymentId`
- `BizRulesImportBootstrapRules`
- `BizRulesLeafSearchNumOfHops`
- `OnlineJavadocPrefix`

IMPORTANT If you are running PolicyCenter business rules on a production server, you must set `BizRulesDeploymentEnabled` to `true` and provide a value for `BizRulesDeploymentId`. If you are running multiple production clusters, the `BizRulesDeploymentID` value for each separate cluster must be unique.

See also

- *Configuration Guide*

Business rule production server configuration

If you use PolicyCenter business rules in a production environment, you must do the following:

- Set configuration parameter `BizRulesDeploymentEnabled` to `true`.
- Set configuration parameter `BizRulesDeploymentId` to the server ID of the production server importing the business rules.

You set the server ID in the `<registry>` element in file `config.xml`, for example:

```
<registry roles="batch, workqueue, scheduler, messaging, startable, special, ui">
  <server serverid= "server_id" roles="batch,workqueue"/>
</registry>
```

Set the value of *`server_id`* to the value of `BizRulesDeploymentId`.

See also

- "Understanding the configuration <registry> element" on page 60
- *Configuration Guide*

Business rule versioning

PolicyCenter associates a version number and a status with each individual business rule. You see this information in the **Business Rules** screen, in the **Version** field. PolicyCenter manages the information in the **Version** field, you cannot change it yourself. If you have multiple versions of a rule, the **Version** field becomes a drop-down list, from which you can select a specific version of a rule to view.

Each version number starts at 0 and increments by 1 for each deployed version of the business rules. If you edit a business rule, PolicyCenter adds a plus sign (+) to the version number of the rule that you are editing, for example, 0+. Along with the version number, each business rule shows its state in parenthesis next to the version number, for example 0+ (Draft). Each version of a business rule is always in one of the following states:

- Draft
- Staged
- Approved

At the creation of a new rule version, PolicyCenter generates a work-in-progress rule with an initial status of Draft. This draft rule contains the same data values as the parent rule version from which it was made.

The work-in-progress rule must go through the three states of Draft, Staged, and Approved before it is possible to deploy the new rule version. After these three stages are complete, and after you deploy the rule, PolicyCenter assigns an updated version number to the rule. This version of the rule becomes the latest running version of the rule that PolicyCenter evaluates at runtime.

The version of a rule that PolicyCenter evaluates at runtime contains the word Evaluated after the version number. The Evaluated version of a rule is always the latest iteration of a rule that can run in a specific environment.

Rule versioning in multi-cluster production environments

PolicyCenter associates the `BizRulesDeploymentId` value (name) with a rule during its deployment. Thus, if you deploy the rule in one production cluster, the name appears along with the rule version number in other production clusters to identify that the rule was deployed in a particular production cluster. For example, if the value of `BizRulesDeploymentId` is GW100-A, the rule version appears as 1.GW100-A in all other production clusters.

Rules for deleting a business rule version

PolicyCenter associates a version number and a status (state) with each individual business rule. Whether it is possible to delete any given version of a rule depends on the state of the business rule. The following table lists the rules for deleting a rule version.

Rule version state	Rule deletion action
Deployed	It is not possible delete a rule version after its deployment
Approved or **Staged**	Clicking **Delete** deletes all rule versions down to the last deployed rule version.
Draft – Direct parent deployed	Clicking **Delete Draft** deletes all rule versions down to the last deployed rule version.
Draft – Rule previously staged or approved	Clicking **Delete Draft** deletes the immediate draft version of the rule. After deleting the draft version of the rule, it then becomes possible to delete the staged and approved versions of the rule.

Business rule deployment

In a production PolicyCenter cluster, Guidewire requires that you deploy a rule before PolicyCenter can evaluate the business logic of the rule. Thus, to deploy a business rule is to make that version of the business rule active within PolicyCenter.

Before deployment, PolicyCenter creates a work-in-progress version of the rule. At rule deployment, PolicyCenter assigns a version number to the rule.

A rule must go through the following states before you can deploy the rule:

- Draft
- Staged
- Approved

A rule must be in the approved state before it is possible to deploy the rule. Approving a rule does not deploy the rule. You must actively deploy a rule before PolicyCenter evaluates that rule at runtime.

Rule deployment is distinct from rule import. On completing a rule import, you must deploy any new approved rule versions before PolicyCenter can evaluate the rules.

In general, Guidewire recommends that you use separate cluster environments for each of the following in working with PolicyCenter business rules:

- Development
- Test
- Production

Typically, you deploy a rule version in a production environment only.

IMPORTANT In a production environment, you must set configuration parameter `BizRulesDeploymentEnabled` to `true` and provide a value for configuration parameter `BizRulesDeploymentId`. If you have multiple PolicyCenter production clusters, the `BizRulesDeploymentId` value for each cluster must be unique.

See also

- “Business rule versioning” on page 333
- “Business rule state” on page 334

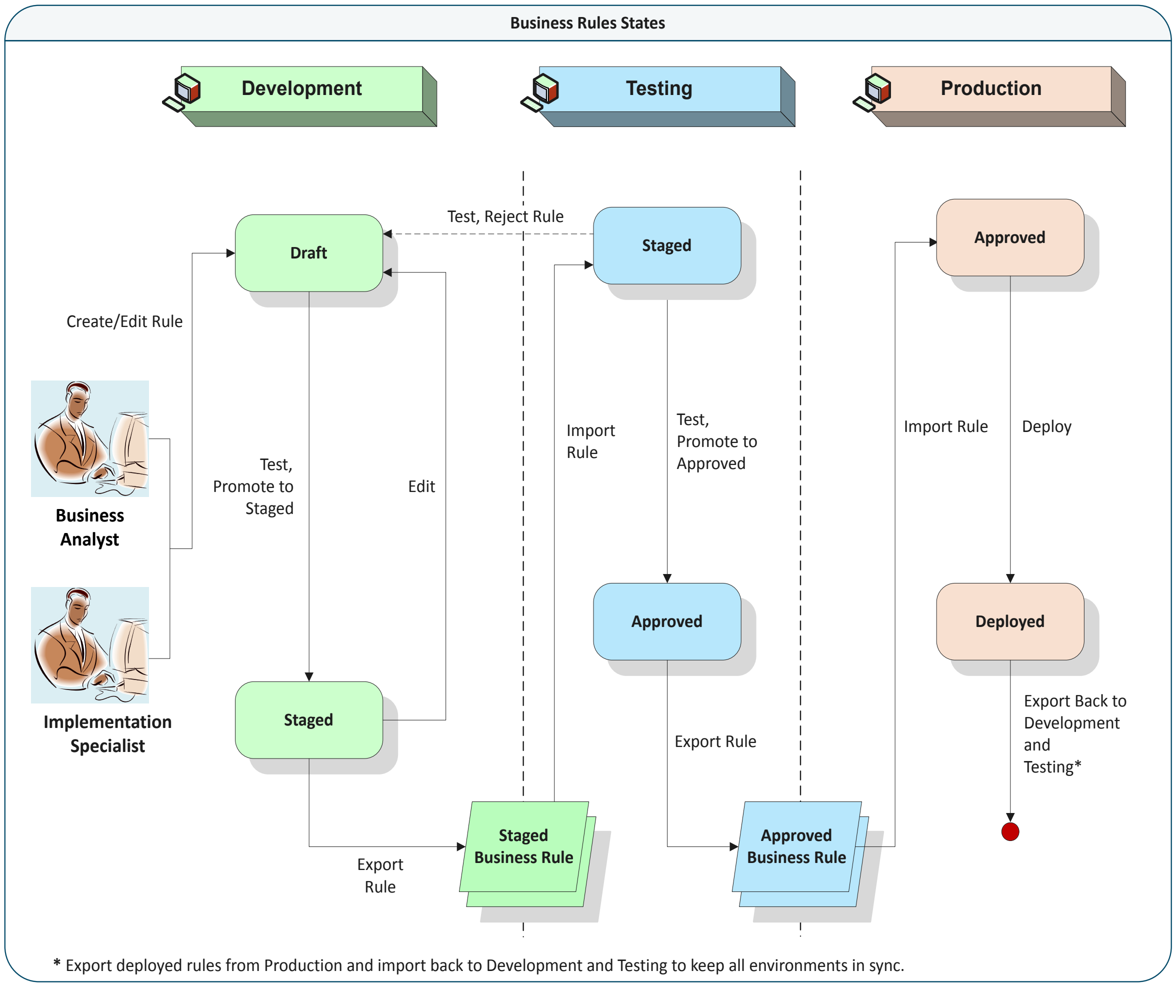
Business rule state

At initial creation, all business rules are in the Draft state. As you begin the process of reviewing, evaluating, and updating a business rules, its state can vary and progress.

The Guidewire PolicyCenter business rules have the following sequence of states.

Draft	PolicyCenter assigns a status of Draft after you save the initial version of the rule. A rule reverts to Draft status whenever the rule undergoes any type of editing. It is not possible export a rule in the Draft state.
Staged	You manually move a rule in the Draft state to the Staged state, after you complete rule editing. Typically, this is the point in the rule lifecycle that you export the rule from the development environment and import the rule into the testing environment.
Approved	You manually move a rule from the Staged state to the Approved state, usually in the testing environment after you complete the rule evaluation phase. Typically, this is the point in the rule lifecycle that you export the rule from the testing environment and import it into the development environment.
Deployed	You manually move a rule from the Approved state to the Deployed state. Typically, this occurs after you import the rule into the production environment.

The following diagram describes the various states associated with a business rule, as it is created, edited, and deployed in development and production environments.



Business rule lifecycle

Whenever you deploy a rule in the production server, Guidewire recommends that you export the rule from the production server and import it back to the development and testing servers. Always make changes on the development server and move the rule to testing then to production. If you always export the deployed rule from production back to development, the version number and rule status on development stays synchronized with the production server.

For example, you create a new rule; the version is 0+. The version stays the same as you move it from development to testing, and then to production. Whenever you deploy the rule on the production server, the version number increases to 1. You export the rule from production back to the development server. Now on both development and production servers, the rule version is 1. On the development, you make changes to version 1. Whenever you import the rule through testing to production and deploy it, the rule version becomes 2.

The following table shows how rule version and status change as you export and import it from development, testing, and production servers.

Action	Development	Testing	Production
1. Create a new rule.	0+ (Draft)		
2. Edit rule and save it. Repeat several times.	0+ (Draft)		
3. Rule is ready for testing. Promote to Staged status.	0+ (Staged)		
4. Export rule and import to testing server.		0+ (Staged)	
5. Rule passes testing. Promote to Approved status.		0+ (Approved)	
6. Export rule and import to production server.			0+ (Approved)
7. Deploy the rule. The version number increases from 0 to 1.			1 (Evaluated)
8. Export rule and import back to development and testing servers.	1	1	
9. Edit rule.	1+ (Draft)		
10. Ready for testing, promote to Staged.	1+ (Staged)		
11. Export and import to testing server.		1+ (Staged)	
12. Testing passed. Promoted to Approved.		1+ (Approved)	
13. Export and import to production.			1+ (Approved)
14. Deploy. The version number increases from 1 to 2.			2 (Evaluated)
15. Export rule and import back to development and testing servers.	2	2	

Guidewire does not require that you move the rule back to development before making changes. However, if you do not export a deployed rule back to the development server, the version number on the development server does not correspond to the version on production. For example, if you never export the rule back to development, the version number stays at 0+ on the development server, and increases each time you deploy it on production.

The following table shows rule version and status change for the same rule, except that you do not export the rule from production to development.

Action	Development	Testing	Production
1. Create a new rule.	0+ (Draft)		
2. Edit rule and save it. Repeat several times.	0+ (Draft)		
3. Rule is ready for testing. Promote to Staged status.	0+ (Staged)		
4. Export rule and import to testing server.		0+ (Staged)	

Action	Develop-ment	Testing	Production
5. Rule passes testing. Promote to Approved status.		0+ (Ap-proved)	
6. Export rule and import to production server.			0+ (Ap-proved)
7. Deploy the rule. The version number increases from 0 to 1.			1 (Evaluat-ed)
8. Rule requires updates. On development server, edit rule and save. Repeat sever-al times.	0+ (Draft)		
9. Ready for testing, promote to Staged.	0+ (Staged)		
10. Export and import to testing server.		0+ (Staged)	
11. Testing passed. Promoted to Approved.		0+ (Ap-proved)	
12. Export and import to production. The version number increases because the rule version currently on the production server is 1 (Evaluated). The rule being imported has changes built on top of that version, so the version becomes 1+. If changes to the rule have also been made on production server, you must re-solve the conflict manually on import.			1+ (Ap-proved)
13. Deploy. The version number increases from 1 to 2.			2 (Evaluat-ed)

Business rule logging

PolicyCenter writes audit information to the console log and to the PolicyCenter log file whenever a rule evaluation triggers a business rule. Unless specified otherwise, PolicyCenter writes the business rule information at the log INFO level for the following logging categories and subcategories:

```
BizRules
  BizRules.Audit
  BizRules.Autocomplete
  BizRules.Compiler
  BizRules.ContextHelp
  BizRules.Export
  BizRules.Import
  BizRules.UI
  BizRule.Validation
```

The business rule information that PolicyCenter logs includes the following:

- The rule context
- The rule ID
- Whether the rule condition evaluates to true or false
- The rule action parameters as a list of name-value pairs, which is empty if the condition evaluates to false

Default logging level

The default logging level for business rules is INFO. It is possible for the standard operation of the business rules to generate a large amount of entries in the business rules logging file. You can reduce the amount and frequency of business rule logging by setting the logging level to a higher threshold, for example, WARN.

To change the default logging level for the business rules, add the following entry to file `log4j2.xml`:

```
log4j.category.BizRules=WARN
```

See also

- "The Set Log Level screen" on page 361

Invalid business rules

Invalid rules during server start

If business rules exist in the database, PolicyCenter validates these rules during the start of the application server. If PolicyCenter finds an invalid rule, it generates an error message in the application log similar to the following:

```
ERROR BizRules.Validation Rule Test Rule has validation error: Could not resolve symbol
for : SomeActivity
```

Guidewire recommends that you review the application log after starting the application server to determine if PolicyCenter reports any rule as invalid.

Invalid rules during rule execution

How PolicyCenter treats an invalid rule depends on the rule state and the server mode.

Rule state	Server mode	Action
Draft	Non-Production	PolicyCenter skips the execution of the invalid rule and continues with rule execution.
Staged, Approved,or Deployed	Non-Production	PolicyCenter generates an exception and stops rule execution.
Deployed	Production	PolicyCenter generates an exception and stops rule execution.

chapter 24

Business rules import and export

You transfer rules between different server environments by exporting the rules from one server environment and importing the rules into the other server environment. Typically, one or more appointed Rules Administrators manage the export and import of business rules between different PolicyCenter server environments.

Overview of business rule export and import

The following flow is a typical business rule development lifecycle:

- Create and develop the draft business rules on a development server.
- Export the business rules to a testing server that replicates the production environment and stage the rules.
- After testing, approve the rules on the staging server.
- Export the business rules to the actual production server and deploy the rules there.

You use the import and export of business rules to move rules between the multiple servers.

During import, PolicyCenter only brings in rules that are new or have changes on top of existing versions. The version number increases whenever you deploy the rule.

Whenever importing rules from one server environment to another, PolicyCenter takes one of the following actions for each rule:

PolicyCenter	Action
Imports the rule	PolicyCenter imports the rule if any of the following are true: • The rule for import is new and does not exist in the importing server environment. • The rule for import is an update to an existing rule in the importing server environment.
Does not import the rule	PolicyCenter does not import the rule into the importing server environment if any of the following are true: • If there is no difference between the existing rule and the rule for import. • If the existing rule is already an update of the rule for import.
Raises a conflict	PolicyCenter raises a conflict if it cannot automatically decide whether to import the rule. This can happen for example, if the existing rule and the rule for import both have updates that conflict. As a concrete example, suppose that you update the rule on the testing server. Then you independently update the rule on the development server. You export the rule from the development server and import it into the testing server. The testing server raises a conflict while trying to import the rule. Within PolicyCenter, you must choose whether to keep the existing rule or take the importing rule. This type of rule conflict can occur also if you update or customize a default business rule and Guidewire later provides an updated version of that rule in an upgrade. In that case, you must choose either to keep one or the other of the updated rules or manage the merge of the two rules.

Rule export does not export utility functions, context objects, or other changes to the business rules plugin. You must propagate changes to Gosu functions or context objects on the development server to the testing and production servers.

About rule version conflicts

During the rule import operation, PolicyCenter raises a rule conflict issue if there is a mismatch in rule versions for any given rule between:

- The existing rule version as defined in the application database
- The importing rule version

The Rule Administrator resolves all rule import conflicts manually through the following business rules screens:

- **Import/Export Status** screen
- **Complete Import** screen
- **Compare Rules** screen

To access the **Import/Export Status** screen, navigate to the following location in Guidewire PolicyCenter:

Administration→Business Settings→Business Rules→Import/Export Status

The other screens open from the **Import/Export Status** screen.

IMPORTANT The user who is responsible for resolving rule conflicts, either in a production environment or non-production environment, must have the necessary rule edit permission.

Rule conflicts in production systems

If you follow the recommended development, test, and production process for business rules, there is little likelihood of rule conflicts while importing rules into a production system. However, Guidewire does not programmatically enforce the recommended process for business rules. Thus, if you do not follow the recommend business rules process, you can experience rule conflicts.

For example, suppose that you modify a business rule on a production system but do not export your changes as Guidewire recommends. A conflict then occurs if you import a version of that rule into the production system that is not based on the modified rule in the production environment. To resolve this issue, you need to do one of the following:

- A user with rule edit permissions selects either the existing rule or the importing rule during the rule import into the production environment.
- A user exports the modified rule from the production environment and then resolve the conflict in a non-production environment. The import of this rule into the production environment does not cause a conflict.

IMPORTANT The user who is responsible for resolving rule conflicts, either in a production environment or non-production environment, must have the necessary rule edit permission.

Business rule import failure

If, for some reason, PolicyCenter detects an error with a rule import operation, it sets the **Status** value for that import operation to **Failure** in the **Import/Export Status** screen. Click the **Failure** link to open the **Rule Import/Export Failure Reason** screen for more information on the reason for the import failure.

Source and target data models and rule export and import

The business rules data model in the source and target systems must match. Otherwise, the rule import into the target system fails. Ensure that the business rules data model of the source and target systems is the same before exporting any business rule. If necessary:

- Modify the business rules data model of one system so that it matches the business rules data model of the other system.
- Regenerate the business rules export file.
- Repeat the business rules import operation.

Export business rules from Guidewire PolicyCenter

About this task

It is possible to export business rules from PolicyCenter in multiple ways from the **Underwriting Rules** screen. To export a rules, it must be in one of the following states: Staged, Approved, or Deployed. It is not possible export a rule in the Draft state.

Procedure

1. Log into Guidewire PolicyCenter using an administrative account.
2. Navigate to the following screen:

   Administration→Business Settings→Business Rules→Underwriting Rules
3. From the **More** drop-down list, chose one of the following options:

Export Selected	Exports only the selected business rules. PolicyCenter exports the selected rules from the currently active screen only.
Export All	Exports all business rules visible on all pages. PolicyCenter does not export rules hidden because of filters. Nor does PolicyCenter export rules in the Draft state that have never been deployed. If a rule is in the Draft state and has one or more previously deployed versions, PolicyCenter exports the last deployed version.

   Selecting one of these options opens the **Import/Export Status** screen.

See also

- "Import business rules into Guidewire PolicyCenter" on page 341
- "The Import /Export Status screen" on page 342

Import business rules into Guidewire PolicyCenter

Before you begin

You can only import a business rule into Guidewire PolicyCenter that was previously exported from Guidewire PolicyCenter.

Data lookup tables are exported and imported separately from business rules. You must import the data lookup tables before importing the business rules that use them. See .

Procedure

1. Log into Guidewire PolicyCenter using an administrative account.
2. Navigate to the following administrative screen:

   Administration→Business Settings→Business Rules
3. Choose one of the following options:

Import rules from...	Import option
Underwriting Rules screen	From the **More** drop-down list, select **Import from File**.
Import/Export Status screen	Click **Import from File**.

Selecting one of these options opens the **Import/Export Status** screen.

See also

- “Export business rules from Guidewire PolicyCenter” on page 341
- “The Import /Export Status screen” on page 342
- “About the business rules export file” on page 342

The Import /Export Status screen

After you begin the import or export of business rules, PolicyCenter opens the **Import/Export Status** screen automatically. This screen shows progress and status information for each import or export session initiated by any PolicyCenter rule administrator. To update the information on this screen, click **Refresh**.

You can also initiate a rule import operation from this screen by clicking **Import from File**.

The Imports table

The **Imports** summary table contains information about each rule import operation, including the following:

- The user who initiated the import operation
- The start time of the import operation
- The source file for the import operation
- The status of the import operation
- A **Review** button if there are no rule conflicts in this import operation
- A **Complete Import** button if this import operation generated rule conflicts between the import and existing rules

The Exports table

The **Exports** summary table contains information about each rule export operation, including the following:

- The user who initiated the export operation
- The start time of the export operation
- The number of rules in the export file
- A **Cancel** button that is visible only during the export operation
- A **Download** button that becomes visible after the export operation completes

See also

- “About the business rules export file” on page 342
- “The Review Import and Complete Import screens” on page 343

About the business rules export file

The **Import/Export Status** business rule screen shows the progress of a business rules export operation. After export completes, click **Download** to save exported rules to the file system.

PolicyCenter writes the rules for export to a file with a `.gwrules` extension. This file is in binary format. It is not possible to view or edit this file outside of Guidewire PolicyCenter. The primary use for this file is to provide the ability to export business rules from one PolicyCenter server environment and import those rules into another PolicyCenter server environment.

See also

- "Export business rules from Guidewire PolicyCenter" on page 341
- "Import business rules into Guidewire PolicyCenter" on page 341
- "The Compare Rules screen" on page 345

The Review Import and Complete Import screens

The business rules **Import/Export Status** screen displays information about each business rule import operation. The functionality changes on the screen depending on whether a rule conflict exists in an import operation.

Rule conflicts?	Functionality
No	If there are no issues to resolve for a given rule import operation, PolicyCenter displays a **Review** button in the **Imports** table for that rule. Clicking **Review** opens the **Review Import** screen.
Yes	If there are pending rule conflicts to resolve for a given rule import operation, PolicyCenter displays a **Complete Import** button in the **Imports** table for that rule. Clicking **Complete Import** opens the **Complete Import** screen.

The **Review Import** and **Complete Import** screens are identical except that the functionality changes depending on whether you are reviewing a rule import operation or resolving a rule import conflict.

The **Review Import / Complete Import** screen consists of the following distinct sections:

- "Rule Import: The Disposition table" on page 343
- "Rule Import: The Manage Synchronization table" on page 344

To access the business rules **Import/Export Status** screen, navigate to the following location within Guidewire PolicyCenter:

Administration→Business Settings→Business Rules→Import/Export Status

Necessary rule permissions

It is important to understand that a user who has only the rule import permission can import rules only. That user cannot resolve any rule conflicts that occur during the rule import. Solving a rule import conflict typically requires the creation of a new rule version. Thus, resolving rule conflicts requires the rule edit permission.

Guidewire separates these two functions (rule import and rule conflict resolution) as a security feature. For the same user to perform both of these functions, that user must have both the import rule and edit rule permissions.

If a user with the edit rule permission only attempts to take certain actions in the **Complete Import** or the **Compare Rules** screen, PolicyCenter displays an error message.

Rule Import: The Disposition table

Clicking **Review** or **Complete Import** in the business rules **Export Status** screen opens either the **Review Import** or the **Complete Import** screen for that rule import operation. These two screens are identical in layout. At the top of the screen is **Disposition** table that provides summary information about the rules in the rule import file. The following list describes the various columns in the table.

Column	Description
Outstanding	The subcategories under **Outstanding** have the following meanings: • **New Rule** – Number of rules in the import file that have no equivalent on the importing server. • **New Version** – Number of rules in the import file that are newer versions of rules on the importing server. • **Rule Deployment** – Number of rules being imported that have a deployed version. • **Version Conflict** – Number of rules for which there are version conflicts between the importing rules and the existing rules on the importing server.

Column	Description
	Note: If there are no rule conflicts for a given import operation, all subcategories for that operation display zero (0).
Imported	Number of rules PolicyCenter imported from the import file.
Discarded	Number of rules discarded by user action.
Applied Edited	Number of rules that the Rule Administrator edited and saved.
No Change	Number of rules that require no additional action. These rules exist in the importing server already and do not require updating. For example, the rule in the import file is an earlier version of the rule version on the importing server.

To access the **Import/Export Status** screen, navigate to the following location in Guidewire PolicyCenter:

Administration→Business Settings→Business Rules→Import/Export Status

Rule Import: The Manage Synchronization table

Clicking **Review** or **Complete Import** in the **Import/Export Status** screen opens either the **Review Import** or the **Complete Import** screen for that rule import operation. These two screens are identical in layout. At the bottom of the screen is **Manage Synchronization** table that provides summary information about each rule in the rule import operation.

If you undertake an action to resolve a rule import conflict, you must save and apply your change. See "Resolve rule import conflicts" on page 345 for details.

The following list describes the various columns in the table.

Column	Description
Rule Name	Name of the rule. PolicyCenter adds a warning next to the rule name if the importing rule name duplicates the name of a differently configured rule on the importing server.
Status	Current status of the rule. Some examples are: • **New Rule** – This rule does not exist on the importing server. • **New Rule Version** – This version of the rule does not exist on the importing server. • **No Change** – The existing rule is a more updated version of the importing rule. • **Rule Deployment** – The existing version of the rule needs deployment to match the rule version of the importing rule. • **Versions conflict** – There are conflicts between the existing and importing versions of the rule that you must manage manually. Depending on the rule status, it is possible for the list of possible actions in the **Available Actions** column to change.
Available Actions	If there is no import conflicts with the given rule, there are no actions available in this column. If there is a conflict between the importing and existing rule versions, the following actions are available: • **Existing Version** – Keep the existing rule on the importing server and discard the imported rule. • **Importing Version** – Keep the importing rule version and discard the existing rule version on the importing server. • **Compare** – Click to open the **Compare Rules** screen. Use this screen to compare the two rules side-by-side to view their similarities and differences and determine the course of action to take.
Existing Version	The version of the rule that exists in the importing server. Click to access a read-only view of the rule.
Importing Version	The version of the rule in the import file. Click to access a read-only view of the rule.

In this table, there is an additional column with no heading just to the left of the **Rule Name** column. An icon (with a gear) in this column indicates that the PolicyCenter business rules editor does not manage this rule. As the rule is external to the editor, it is not possible to edit the rule in most respects.

To access the **Import/Export Status** screen, navigate to the following location in Guidewire PolicyCenter:

Administration→Business Settings→Business Rules→Import/Export Status

Business rule validation errors

If there is at least one rule validation error in the business rules **Complete Import** screen, PolicyCenter inserts an exclamation icon (!) to the immediate left of the name of the invalid rule. If you hover the cursor over the icon, PolicyCenter shows additional information in the tooltip message.

PolicyCenter shows ... to the immediate left of the rule name if the validation work queue has not yet run and there are pending validations.

The Compare Rules screen

PolicyCenter opens the **Compare Rules** screen if you click a **Compare** link in **Complete Import** screen. The **Compare** link appears if there is a conflict between an existing rule and a rule that you are importing.

The **Compare Rules** screen creates a side-by-side table view to make it easier to compare the details of the two rules. PolicyCenter highlights the fields that are different between the two rule definitions.

After you review each of the rules, choose one of the following actions.

Keep Existing Version	Chose to retain the currently existing rule and discard the importing rule. If you select this option, PolicyCenter reopens the **Complete Import** screen with the **Existing Version** radio button selected for the rule under **Available Actions**.
Replace with Importing Version	Chose to accept the importing rule and discard the existing rule. If you select this option, PolicyCenter reopens the **Complete Import** screen with the **Importing Version** radio button selected for the rule under **Available Actions**.
Edit	Select one of the following from the **Edit** drop-down list to modify either the existing or importing rule: • **Existing Version** • **Importing Version** If you choose to edit a rule version, PolicyCenter opens the rule in edit mode. Editing a rule creates a new Draft version of the rule. If you save the edited rule, it becomes the resolved rule version for import completion. After making this update, you can no longer select either the importing or existing rule in the **Complete Import** screen. The status of the rule in the **Complete Import** screen changes to **Edited Version**. If you select the rule and apply your change, the **Applied Edited** field increments by one (1). PolicyCenter disables the edit functionality if the existing rule version is in Draft mode.

See also

• "Resolve rule import conflicts" on page 345

Resolve rule import conflicts

About this task

To resolve rule version conflicts that occurs during a rule import process, perform the following sequence of steps.

IMPORTANT The user who is responsible for resolving rule conflicts, either in a production environment or non-production environment, must have the necessary rule edit permission.

Procedure

1. Navigate to the following location in Guidewire PolicyCenter:
   Administration→Business Settings→Business Rules→Import/Export Status
2. For each rule import that has a status other than **Completed**, click **Complete Import**.
   The **Complete Import** screen opens.
3. Review each rule that shows an import conflict in the **Complete Import** screen. Do one of the following:
   a. Click the **Existing Version** or the **Importing Version** link to see the details for that rule version.

b. Click the **Compare** link to open the **Compare Rules** screen in which you can view the two rule versions in a side-by-side table.

4. (Optional) After reviewing the two rules in the **Compare Rules** screen, do one of the following:
   - Click **Keep Existing Version** to accept the existing rule on the import server with no change.
   - Click **Replace with Importing Version** to accept the import rule with no change.
   - Select an **Edit** option to open an editable view of either the existing or importing version of the rule.
   - **Note:** PolicyCenter disables the edit functionality if the existing rule version is in Draft mode.
5. (Optional) Make your edits the **Compare Rules** screen, then click **Save Edited Version**.

   If you edit either of the rule versions in the **Compare Rules** screen, that version of the rule becomes the resolved version of the rule. Thereafter, you cannot ever use the importing or existing version of the rule to resolve the rule conflict.

   PolicyCenter reopens the **Complete Import** screen.
6. In the **Complete Import** screen, select the resolution type by selecting a radio button in the **Available Actions** cell for the rule of interest. Your choices are:
   - Use the existing rule version.
   - Use the importing rule version.
   - Use the new, edited, version of the rule.
7. Select the rule for update by selecting the check box to the left of the rule name.
8. Save and apply your changes by clicking one of the following function buttons:

Import Selected	Enabled if you select one or more rules. It must be possible to import these rules.
Discard Selected	Enabled if you select one or more rules.
Import All Remaining	Enabled if there are no remaining unresolved conflicts. It must be possible to import these rules.
Discard All Remaining	Enabled if you select one or more rules.

9. Repeat these steps as necessary to resolve all rule conflicts in the rule import operation.

See also

- "The Compare Rules screen" on page 345

Automatic import of business rules at server startup

After the initial PolicyCenter installation, the database is empty of data. The first time that you start the PolicyCenter application server after installation, PolicyCenter upgrades the database and populates it with data. As part of this initial database upgrade, it is possible to automatically load data from the following Studio **Project** folder into the PolicyCenter database:

configuration→config→import→bizrules

Configuration parameter `BizRulesImportBootstrapRules` in `config.xml` controls whether PolicyCenter imports the business rules in folder **bizrules** automatically at initial server start, if you have an empty database.

IMPORTANT Guidewire recommends that you set this parameter to `true` in an environment in which you plan to do initial rule review or rule development only.

See also

- "Business rules import at initial server startup" on page 299

Import PolicyCenter business rules at server startup

About this task

It is possible to automatically import any business rules that exist in the PolicyCenter `bizrules` directory at initial server startup.

Procedure

1. Set configuration parameter `BizRulesImportBootstrapRules` in `config.xml` to `true`.
2. Place the business rules to import in the following location in PolicyCenter Studio:

   configuration→config→import→bizrules
3. With an empty database, start the PolicyCenter application server.

   PolicyCenter populates the database with the business rules in the `bizrules` directory.
4. Reset configuration parameter `BizRulesImportBootstrapRules` to `false`.

See also

- "About the import directory" on page 298
- *Configuration Guide*

Correct duplicate BizRuleDeploymentId values

About this task

If you attempt to import a set of business rules that has a `BizRulesDeploymentId` value that is the same as the current PolicyCenter cluster, the rule import fails. For example, this happens if a development cluster has the same `BizRulesDeploymentId` value as the production cluster into which you are importing the files. To avoid this scenario, Guidewire recommends that each PolicyCenter sever cluster have its own unique value for `BizRulesDeploymentId`.

Note: You set the value of `BizRulesDeploymentId` in `config.xml`.

Correcting this issue is a multi-step process:

Procedure

1. Temporarily, start the application server with the value of `BizRulesDeploymentEnabled` set to `false`.

   In this case, PolicyCenter ignores the value of `BizRulesDeploymentId` and import validation does not stop the import of the file.
2. After the rule import succeeds, restart the application server with the value of `BizRulesDeploymentEnabled` set to `true`.

See also

- *Configuration Guide*

Importing rules after a data model change

During an export of business rules, PolicyCenter records the data model of the exporting environment and creates a checksum for that data model. PolicyCenter then adds this checksum to the business rules export file. On import of the exported file, PolicyCenter compares the checksum of the export environment with the checksum of the import environment. PolicyCenter uses the checksum comparison to ensure that the data model of the exporting and importing environments match:

- If the two checksums match, PolicyCenter imports the exported rules.
- If the two checksums do not match, the import process fails.

The process of continually matching data models between export and import environments can cause problems with rule import during the PolicyCenter development phase, in particular. During PolicyCenter development, the data model often changes frequently Thus, you can export the business rules, modify the data model, and then fail at the attempt to import the business rules as the data model checksums of the export and import environments are different.

To minimize the need to regenerate the rule export file after a data model change, Guidewire provides a system property that disables checksum verification on business rule import.

System property VerifyRuleImportDataModelCheckSum

System property `VerifyRuleImportDataModelCheckSum` can be one of the following Boolean values.

`true`	PolicyCenter enforces verification of the data model during rule import. The default is `true`.
`false`	PolicyCenter does not enforce verification of the data model during rule import.

System property `VerifyRuleImportDataModelCheckSum` is only valid on a PolicyCenter server under the following circumstances:

- The server mode is either `DEV` or `TEST`. The server mode cannot be PROD (production).
- The value of configuration parameter `BizRulesDeploymentEnabled` is `false`.

Setting system property VerifyRuleImportDataModelCheckSum

You can set system property `VerifyRuleImportDataModelCheckSum` in the following ways:

- In PolicyCenter Studio, in the server JVM options.
- From a PolicyCenter command prompt, as you start the application server.

Set VerifyRuleImportDataModelCheckSum from a command prompt

Use system property `VerifyRuleImportDataModelCheckSum` to disable verification of the data model during rule import after a data model change.

About this task

The syntax for setting a system property from a command prompt can change depending on the type of the application server. See "Setting JVM options in PolicyCenter" on page 66 for more information.

Procedure

1. Stop the PolicyCenter server on which you want to set the system parameter, if it is currently running.
2. Open a command prompt on the server.
3. Start the server using the following command:

   ```
   gwb runServer -DVerifyRuleImportDataModelCheckSum=value
   ```

   Parameter `value` can take the following values.

`true`	(Default) Enable verification of data model on rule import.
`false`	Disable verification of data model on rule import.

Set VerifyRuleImportDataModelCheckSum in Studio

Use system property `VerifyRuleImportDataModelCheckSum` to disable verification of the data model during rule import after a data model change.

Procedure

1. Log into PolicyCenter Studio.
2. From the Studio toolbar, navigate to the following location:

   Run→Edit Configuration...
3. In the **Run/Debug Configuration** dialog that opens, expand the **Application** tree view.
4. Select the server instance to which you want to apply the system parameter.
5. In the **VM options** field for that server, add the following text:

   `-DVerifyRuleImportDataModelCheckSum=`*`value`*

   Parameter *`value`* can take the following values.

`true`	(Default) Enable verification of data model on rule import.
`false`	Disable verification of data model on rule import.

6. Click **Apply**, then **OK**.
7. Restart the selected server instance.

part 7

Administration tools

chapter 25

Server tools

Guidewire provides server tools to assist you with certain server and database administration tasks.

See also

- “Internal tools” on page 417

Accessing the PolicyCenter server tools

You must have the `internaltools` permission to access the **Server Tools** screens. If the value of parameter `EnableInternalDebugTools` in `config.xml` is `true` and the server is running in development mode, all users have access to the **Server Tools** screens.

To access the **Server Tools** screens, press `ALT+SHIFT+T` on any PolicyCenter screen after you log into the application.

See also

- “Server modes” on page 83

The Batch Process Info screen

Use the Server Tools **Batch Process Info** screen to view information about PolicyCenter batch processes, including writer threads for work queues. Use the drop-down on the **Batch Process Info** screen to filter the batch process list. This drop-down contains the following options.

Any	Shows all batch processing types.
Schedulable	Shows batch prossing types marked as `Schedulable` in the `BatchProcesstype` typelist. You define a schedule for batch processes, including writer processes for work queues, in file `scheduler-config.xml`.
Runnable	Shows batch procesing types marked as `APIRunable` in the `BatchProcesstype` typelist. It is possible to run these types of batch processes using APIs.

From this screen, you can perform the following actions.

Refresh	Click to update the information in the **Processes** table.
Download	Click to download some of the information on this screen as a HTML report. See “Download a batch process history report” on page 355 for details.

Suspend Scheduler	Stops the batch processing scheduler that triggers the execution of the batch processes. After stopping the scheduled, you can restart it using this same button.

The **Batch Process Info** screen contains the following areas or tabs that contain batch process information.

Processes	Lists all the available batch processing types along with information about each individual batch process. It is also possible to run certain actions on a selected batch processing type from this screen. See "Processes table columns" on page 354 for more information.
Chart	Shows the execution time in seconds and the number of operations performed by the batch process over time in a graphical format.
History	Shows records of past runs of the selected batch process in tabular form.

Note: It is possible to run writers for work queues either from the **Work Queue info** screen or from the **Batch Process Info** screen.

See also

- "Administering PolicyCenter processes" on page 113
- "Work queues and batch processes, a reference" on page 130
- "Processes table columns" on page 354
- "Chart and History tabs" on page 355
- "The Work Queue Info screen" on page 356

Processes table columns

The **Processes** area of the Server Tools **Batch Process Info** screen contains a number of columns. The column labels have the following meaning:

Column	Description
Batch Process	Name of the batch processing type.
Description	Description of the batch processing type.
Action	Actions that you can perform on the selected batch processing type. These actions include: • **Run** – Runs a batch process. The **Run** button is active for all batch process types that belong to the `BatchProcessTypeUsage` category `UIRunnable`. • **Stop** – Stops an actively running batch process. • **Download History** – Downloads a batch process history report for the selected batch process in HTML format. See "Download a batch process history report" on page 355 for more information. You cannot start multiple runs of non-exclusive custom batch processes from the **Batch Process Info** screen. Instead, you must use the `maintenance_tools` command to start multiple runs of non-exclusive custom batch processes.
Last Run	Date on which this batch processing type last run.
Last Run Status	Completion status from the last run of this batch processing type. This field shows as **Failed** or **Interrupted** if the batch process `OperationsFailedReasons` is non-null.
Next Scheduled Run	Date of the next scheduled run for this batch processing type.
Schedule	Scheduling actions that you can perform on the selected batch processing type. These actions include: • **Stop** – Disables the scheduled runs for the selected batch processing type. • **Start** - Enables the scheduled runs for the selected batch processing type.
Cron-S M H DOM M DOW	Column header stands for seconds, minutes, hours, days of month, month, and day of week.

See also

- See "Work queues and batch processes, a reference" on page 130 to determine if it is possible to run multiple instances of a batch processing type.
- See "maintenance_tools command" on page 426 for details of how to start multiple batch processes of the same type.
- See the *Integration Guide* for a discussion of the meaning of the `Exclusive` property on a batch process.

Chart and History tabs

The Server Tools **Batch Process Info** screen contains **Chart** and **History** tabs at the bottom of the screen. Select a batch processing type to view its chart or history.

The **Chart** tab shows the execution time in seconds and the number of operations performed by the batch process over time in a graphical format. The **History** tab includes a table of records of past runs of the selected batch process in tabular form.

The **History** table includes the following information:

Column	Description
Start Requested	The time at which PolicyCenter received the request to start the process.
Started	The time that a batch process started.
Completed	The time that a batch process completed.
Scheduled	Yes indicates that the start request was the result of the regular execution of a schedule. No indicates that a user made the request manually.
Server	Server on which the start request was made. The server on which the request was made is not necessarily the server on which PolicyCenter executed the batch process.
Description	Optional description.
Ops	For batch processes that are work queue writers, the value of **Ops** (operations) is the number of work items that the work queue processed. This number includes work items that failed.
Failed	For batch processes that are work queue writers, the value of **Failed** is a counter that PolicyCenter increments each time work item processing encounters an exception. It is possible for a work queue to attempt to process a failed work item multiple times. Therefore, the **Failed** and **Ops** numbers do 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.

See also

- "The work queue scheduler" on page 120

Download a batch process history report

Procedure

1. Navigate to the Server Tools **Batch Process Info** page.
2. Select a batch process in the **Processes** table.
3. Click **Download History** in the **Actions** column of the row for that particular batch process.
4. Select the date rage for the records that you want to download.
5. Click **Complete Download**.
6. Select the location for the local file download and click **OK**.
7. Unzip the download file into a local directory.
8. Find and double-click `index.html` to open the report.

See also

- “The Batch Process Info screen” on page 353
- “maintenance_tools command” on page 426

The Work Queue Info screen

Use the Server Tools **Work Queue Info** screen to control and view information associated with work queues. From this screen, 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.

The **Work Queue Info** screen contains multiple tables that contain work queue information. These tables include:

- “Work Queue table columns” on page 356 table
- “Item Statistics tabs and columns” on page 357 table

Besides the work queue information available on the **Work Queue Info** screen, it is possible to download a number of specialized reports from this screen as well.

Note: It is possible to run writers for work queues either from the **Work Queue info** screen or from the **Batch Process Info** screen.

See also

- “Work queues and batch processes, a reference” on page 130
- “Work Queue table columns” on page 356
- “Item Statistics tabs and columns” on page 357
- “Work queue reports” on page 359
- “The Batch Process Info screen” on page 353

Work Queue table columns

The **Work Queue** area of the Server Tools **Work Queue Info** screen includes a table containing information and functionality related to work queues. The table column headings have the following meaning:

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.
Executors Running	Number of workers processing the work queue.
Cluster-wide State	Work queue status, for example, started or stopped.
Writer Status	Status of the writers.
Actions	Actions that you can perform on the selected work queue. These actions include: • **Run Writer** – Launches the writer to write work items for the work queue. • **Notify** – Wakes workers by notifying the executor that there are items to process. • **Stop** – Stops the selected work queue. • **Restart** – Restarts the selected work queue. • **Download History** – Downloads the historical instrumentation data for the work queue, in CSV format.

See also

- See “Worker task management” on page 126 for a discussion of the executor function.
- See “The Work Queue History report” on page 360 for more information on the work queue history report.

Item Statistics tabs and columns

The **Item Statistics** area of the Server Tools **Work Queue Info** screen provides information on work items related to the work queue selected in **Work Queue** table. This region has three tabs the contain a number of tables.

Tab	For more information
By Writers	“The By Writers tab” on page 357
By Executors	“The By Executors tab” on page 357
Work Item	“The Work Items tab” on page 358

The By Writers tab

The **By Writers** tab on the Server tools **Work Queue Info** screen contains item counts generated by the writer associated with the work queue selected in the **Work Queue** table. Each row represents one wake period for the writer. These table column headers have the following meanings.

Column	Description
Process ID	ID for the writer process.
Item Creation Time	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.
Server	Name of the server running the work queue.
Scheduled	Yes indicates that the start request was the result of the regular execution of a schedule. No indicates that a user made the request manually.
Number of Items	Total work items in the queue regardless of status.
Worker End Time	Time at which the last worker completed the last item in the work queue.
Execution Time	Time, in minutes, since the start of the process (the execution time so far).
Available	Total number of available items in the queue.
Checked Out	Number of items checked out by workers for processing.
Succeeded	Number of items that completed successfully.
Failed	Number of items that failed.

The By Executors tab

The **By Executors** tab on the Server Tools **Work Queue Info** screen lists active executors. The table column headers have the following meanings:

Column	Description
Hostname	Server on which the executor is running.
Max. Number of Workers	Maximum number of workers available to the executor.
Processed Items	Number of items processed by the workers.
Exceptions	Number of exceptions encountered during processing.

Column	Description
Failed items	Number of work items that failed processing.
Status	Status of the executor, for example, running.
Started	Timestamp of the start of the executor.
Up For	Length of time since the start of the executor.

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 worker succeeded in the processing of the work items.
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 adopted for processing.
Failed items	The number of work items that failed.
Skipped items	The number of items that the process skipped.
Started	Timestamp of the start of the task.
Ended	Timestamp of the end of completion of the task.
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

The **Work Items** tab on the Server Tools **Work Queue Info** screen lists work items. The table column headers have the following meanings.

Column	Description
ID	Unique identifier of the work item.
Create time	Timestamp of the work item creation time.
Update time	Timestamp of the last update to the work item.
Available at	Timestamp of when the work item is available processing. This value is null for failed work items.
Server	Server that processed the work item.
Writer	Writer that created the work item.
Attempts	How many attempts a worker has made to process the item.
Activity ID	ID of the activity involved. This field is only visible if you select the Activity Escalation work queue.
Subject	Subject of the activity involved. This field is only visible if you select the Activity Escalation work queue.

PolicyCenter shows data on this screen during the active execution of database checks only.

Work queue reports

Use the Server Tools **Work Queue Info** screen to generate and download multiple report types of work queue data. You can use the data from these reports to calculate different kinds of secondary data, for example, work queue efficiency.

The Work Queue report

The **Work Queue** report available from the Server Tools **Work Queue Info** screen contains the following linked HTML files:

- Files that include a summary of the work queues
- Files that include detailed information for specific work queues

The report provides data for each worker by thread and by host, such as:

- Last wake up time
- Start and end times for the worker
- How long since the start of the worker
- How many items the worker processed
- Up-time (cumulative, average, and maximum)
- Execution time (cumulative, average, and maximum)
- Items processed for each thread (cumulative, average, and maximum)
- Throughput (items processed per minute of execution) per thread and per host

For each report, you can specify the following:

- The maximum number of writer runs, executors, and batches for each worker
- The number of hours for which to generate item distribution data in the report

  Note: Internally, Guidewire sets the allowable maximum number of writer runs to show in the report at 150.

The Work Queue Runs view

The Work Queue report available from the Server Tools **Work Queue Info** screen 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.

See also

- "Work queue reports" on page 359
- "Download a work queue report" on page 359

Download a work queue report

Procedure

1. Navigate to the Server Tools **Work Queue Info** page.
2. Click **Download** on the top-level menu.
3. Specify the parameters for the report.
4. Click **Complete Download**.
5. Select the location for the local file download and click **OK**.
6. Unzip the download file and double-click `index.html` to open the Work Queue report.
7. Click the **Work Queue Runs** link at the bottom of the Work Queue report to open the Queue Runs view.

See also

- "Work queue reports" on page 359
- "The Work Queue report" on page 359

The Work Queue Raw Data report

The **Work Queue Raw Data** report available from the Server Tools **Work Queue Info** screen is a set of CSV-formatted files. There is one CSV file for each work queue. The files contain time-sliced raw data from the `ProcessHistory` and `InstrumentedWorkerTask` tables.

You can perform analysis of this data in third-party tools such as Microsoft Excel.

See also

- "Work queue reports" on page 359
- "The Work Queue Raw Data report" on page 360

Download the Work Queue Raw Data report

Procedure

1. Navigate to the Server Tools **Work Queue Info** page.
2. Click **Download Raw Data** on the top-level menu.
3. Select the date range for the records that you want to download.
4. Click **Complete Download**.
5. Select the location for the local file download and click **OK**.
6. Unzip the download file and open the work queue instrumentation reports individually.

See also

- "Work queue reports" on page 359

The Work Queue History report

The Server Tools **Work Queue Info** screen includes the ability to download information on the history of a particular work queue. The report, in CSV format, includes the following information:

- Process ID
- Writer start and end times
- Duration
- Failures by workers

You can clear the instrumentation data for all work queues by running the Work Queue Instrumentation Purge process.

See also

- "Work queue reports" on page 359
- "Download the Work Queue History report" on page 360
- "Work Queue Instrumentation Purge batch process" on page 158

Download the Work Queue History report

Procedure

1. Navigate to the Server Tools **Work Queue Info** page.
2. Select a work queue in the **Work Queue** table.
3. Click **Download History** in the **Actions** column of the row for that particular work queue.
4. Select the date range for the records that you want to download.
5. Click **Complete Download**.
6. Select the location for the local file download and click **OK**.

7. Unzip the download file and open the **Work Queue History** report.

See also

- "Work queue reports" on page 359
- "The Work Queue History report" on page 360

About work queue efficiency

The Server Tools **Work Queue Info** screen provides a wide variety of data, along with types of multiple work queue reports. It is possible to use this information to generate additional types of data. For example, the Work Queue report contains information on the processing time for each item (**Item Processing Time**). Using this data, you can calculate the efficiency of a work queue by dividing the work item processing time by the total active time of a worker.

The Set Log Level screen

Use the Server Tools **Set Log Level** screen to set a specific logging level for the different logging categories. The logging levels that you specify in the **Set Log Level** screen persist until you change them or restart the server.

The **Logger** drop-down presents the following types of logging categories:

Type	Description
Guidewire default	Guidewire provides a number of default PolicyCenter logging categories. You can also see the list of Guidewire logging categories by running the `system_tools` command from a command prompt and adding the `-loggercats` option.
Guidewire internal classes	Guidewire provides a number of logging categories that apply to Guidewire internal classes. These logging categories start with `com.guidewire.*`. These logging categories generally look like a fully qualified class path.
Third-party software	Guidewire applications integrate with certain types of third-party software. The manufactures of this software provide their own logging categories. These logging categories start with `org.*`, for example, `org.apache.*` or `org.eclipse.*`. These logging categories generally look like a fully qualified class path.

Making logging level changes permanent

To make your log level settings permanent, you must edit file `log4j2.xml` and set the log level there. Access this file from the **Project** window in Studio by navigating to **configuration→config→logging**, and then opening `log4j2.xml`.

In the default PolicyCenter configuration, the `log4j2.xml` file does not contain a unique logger definition for each Guidewire logger.

Note: Some log4j loggers do not appear on the **Set Log Level** screen until actually used. This is a standard log4j behavior.

Setting logging levels through system tools

It is also possible to set a logging level through the following system tools command.

```
system_tools -updatelogginglevel logger level
```

The PolicyCenter administration command prompt tools all require that you enter the password of an administrative user for the tool to work. The use of a user name is optional.

To use this command, you must supply the name of a specific logger category (*`logger`*) and the new logging level (*`level`*) for that logger. Use the `system_tools -loggercats` command option to see a list of valid PolicyCenter logger categories.

You must refresh the Server Tools **Set Log Level** screen after using the `system_tools` command to see your changes reflected in that screen.

See also

- "system_tools command" on page 429

The View Logs screen

The Server Tools **View Logs** screen contains the following items:

Field	Purpose
Log File	Use to select the log file to view.
Filter	Use to display the log file lines that contain the specified word or words.
Max Lines to Display	Use to set the maximum number of lines to show on the screen.

By default, PolicyCenter writes log files to `tmp/gwlogs/PolicyCenter/logs/`. To specify a different location in which the **View Logs** screen checks for log files, specify a different value for property `guidewire.logDirectory` in file `log4j2.xml`.

Info pages

The Server Tools **Info Pages** provide information to help manage a PolicyCenter server and database. Guidewire intends these screens 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 screens to review the results of a load operation.

The Configuration screen

The Server Tools **Configuration** screen lists the values of the configuration parameters in your PolicyCenter environment. This screen 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`

You can find these files 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. Guidewire makes the in-memory state available because it is possible to change certain configuration parameters using a web service or JMX APIs after server startup.

The Archive Info screen

Use the Server Tools **Archive Info** screen to view information about any archiving processing taken by PolicyCenter.

> **Note:** the value of configuration parameter `ArchiveEnabled` must be `true` before you can view the **Archive Info** screen.

On the **Archive Info** screen, you can take the following actions.

Refresh	Click to update the information on the **Archive Info** screen. Clicking the **Refresh** button also refreshes the `IArchiveSource` plugin.
Download	Click to download archive information to an HTML report. This report is a summary of the information shown on the **Archive Info** screen.

View Progress	Click to open the **Work Queue Info** screen. From this screen you can view the progress of the **Archive** work queue. You can also start an unscheduled run of the archive work queue from the **Work Queue Info** screen. For more information, see "The Work Queue Info screen" on page 356.
Export Upgrade Info	Click to download and save a `.dat` file that contains information on the database version for a specific archive operation.
Import Upgrade Info	Click to upload a `.dat` file that you previously exported. You must browse for a file (click **Browse**) to upload before the **Import Upgrade Info** button becomes active.
Browse...	Click to open a standard file picker dialog.

The **Archive Info** screen contains the following tables.

Overview	The **Overview** table includes the following: • The number of entities that PolicyCenter archived • The number of items that the archiving process skipped or excluded • The number of items that failed the archiving process The screen information divides the excluded items into the following categories: • Items that PolicyCenter excluded from archiving due to business logic • Items that PolicyCenter excluded from archiving due to a failure Click **Reset** to set the total number of items excluded to zero.
Archive Source Information	The **Archive Source Information** table indicates the last refresh time of the **Archive Info** screen and the `IArchiveSource` plugin. The **Archive Source** Information also shows the availability of the store, retrieve and delete services. PolicyCenter derives these values from the following variables in Gosu class `ArchiveSource`: • `storeStatus` • `retrieveStatus` • `deleteStatus` Possible values for these services include: • **Available** – The service is available. • **Failure** – The last archive, restore, or delete operation failed. • **Manually** – A user has manually flagged the service as unavailable. • **Not configured** – The service is not yet configured. • **Not enabled** – Archiving is not yet enabled. • **Not started** – Archiving is not yet started. • **Queue** – The service is not available. PolicyCenter, however, allows the queueing of user requests.
Archive Summary by Data Model Version	As its name suggests, the **Archive Summary by Datamodel Version** table shows archiving information organized by data model version. For each database version, the table shows the following: • The number of entities PolicyCenter succeeded in archiving. • The number entities that PolicyCenter did not archive due to business logic. • The number of entities that PolicyCenter did not succeed in archiving due to a failure. Each excluded category has a **Reset** button to set the count of excluded entities back to zero. Click a specific version to view additional archive information for that data model version. See "The Archive Summary by Data Model Version table" on page 363 for more information.

See also

- *Application Guide*

The Archive Summary by Data Model Version table

The Server Tools **Archive Info** screen contains an **Archive Summary by Data Model Version** table. This area organizes archive data by data model version. Clicking a specific data model version link opens an **Archive Summary for Data Model (n, n, n, n, n)** screen.

On the **Archive Summary** screen, you can do the following.

Reset counts	Click **Reset** to set the value of **Excluded** or **Failed** items back to zero.
Filter the information by time period	Set a value for **Begin Time** and **End Time**, then click **View**.
Review archived items	Select the **Archived** tab to view a list of items archived with this data model version and for the specified time period.
Review skipped, excluded, and failed items	Select the appropriate tab to view the reason that archiving process skipped or excluded an item from archiving, or the reason the items failed the archiving process. For each reason, you can click **Reset All Items** to reset the count to zero.

The Domain Graph Info screen

The Server Tools **Domain Graph Info** screen includes the following tabs:

- **Graphs**
- **Warnings**

IMPORTANT Guidewire strongly recommends that you review the **Warnings** tab for possible issues every time you change the data model.

The graphs tab

The **Graphs** tab of the Server Tools **Domain Graph Info** screen contains the following items:

- A DOT formatted version of the archive domain graph. The DOT format is a means of showing object relationships in plain text.
- A **Download** button, which if clicked, generates a ZIP file that contains the text version of the archive domain graph and the means to view the graph visually.

The warnings tab

The **Warnings** tab of the Server Tools **Domain Graph Info** screen shows non-fatal violations of the archive domain graph that PolicyCenter detected while starting the server. PolicyCenter provides warnings for these situations rather than preventing the server from starting because it is possible to prevent the erroneous situation through business logic.

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** screen to view errors detected by these checks. Instead, the server reports the error in the application log and prints the archive domain graph in DOT notation.

See also

- *Configuration Guide*

The Consistency Checks screen

Use the Server Tools **Consistency Checks** screen to view and run consistency checks on the PolicyCenter database. The screen consists of two tabs:

- **Run consistency checks**
- **View consistency checks definitions**

The Server Tools **Consistency Checks** screen executes the Database Consistency Check batch process. See “Database Consistency Check work queue” on page 137 for more information.

Note: If the server running a database consistency check fails for some reason, PolicyCenter provides for a graceful recovery. After the server becomes operational again, PolicyCenter starts the check at the place it last left off and completes the check.

The Run Consistency Checks tab

The **Run Consistency Checks** tab on the Server Tools **Consistency Checks Info** screen manages the process of running database consistency check batch processing. From this screen, you can:

- Start, stop, pause, restart, or cancel consistency check batch processing.
- Set the tables or table groups on which to run the consistency checks.
- Set the number of workers to use in the batch processing.
- Set the type of consistency checks to run.
- Review the results of each consistency check batch processing in tabular format.

The **Run Consistency Check** tab contains the following items.

Item	Action
Download all selected	Click to download the consistency check results for all selected consistency check runs in the results table. See "Run a consistency check from PolicyCenter" on page 367 for information on how to run a consistency check.
Delete	Click to delete the results of prior consistency check runs for those rows with a check mark in the results table.
Refresh	Click to update the information on the screen while processing is active.
Run Consistency Checks	Click to submit a batch processing job to perform one or more database consistency checks. This action executes the Database Consistency Check batch process. See "Database Consistency Check work queue" on page 137 for more information. This button is not available if the Database Consistency Check work queue is not active, **See also** • "Run a consistency check from PolicyCenter" on page 367 • "Run a consistency check using system tools" on page 368
Pause/Resume Consistency Checks	Click to pause a currently executing batch processing job. During a pause in processing, PolicyCenter changes the button label to **Resume**. Clicking the button resumes processing execution. If this button is not visible, click the **Refresh** button.
Cancel Consistency Checks	Click to cancel all currently executing batch processes. PolicyCenter displays this button only if there is a currently executing consistency check.
All tables **Specify tables** **Specify table groups**	Click to select **All tables** (default) to run consistency check against all database tables. If you select **Specify tables**, PolicyCenter opens a table picker from which you can select the database tables against which the consistency checks run. You must select at least one table. If you select **Specify table groups**, PolicyCenter opens a table groups picker from which you can select the table groups against which the consistency checks run. You must select at least table group.
Change	Click to change the number of workers used to execute the consistency check. Enter a positive integer value. If you change the number of workers in an active work queue, PolicyCenter stops the existing work queue and restarts it with the new number of workers. This button is not available if the Database Consistency Check work queue is not active,
Check all types?	Click to select **Specify Types** to see the list of available check types from which you can make a selection of

The consistency checks results table

After completing a consistency check, the results table columns have the following meanings.

Column	Description
Download	Click the **Download** icon to download a `ConsistencyCheck`*`Rundate`*`.zip` file that contains the set of database reports generated by this consistency check run.

Column	Description
Download Errors	Click the **Download Errors** icon to download a `ConsistencyCheckRunErrorsOnly`*`RunDate`*`.zip` file that contains the consistency check log file and stack trace. PolicyCenter displays this column only if there are SQL errors in the database consistency check. See "Correct a consistency check SQL failure" on page 368 for more information.
View	Click the **View** icon to open a pop-up from which you can view the same reports contained in the `ConsistencyCheck`*`Rundate`*`.zip` file, after you supply your user credentials. PolicyCenter displays this column in test and development mode only. The column is not visible in production mode.
Delete	Click the **Delete** icon to remove a consistency check row from the table.
Rerun	Rerun a consistency check that has a SQL error. PolicyCenter displays this button only if there are SQL errors in the database consistency check. See "Correct a consistency check SQL failure" on page 368 for more information.
Description	List of tables against which the consistency checks ran.
With Errors	Number of errors encountered by the consistency checks run.
Total Checks	Total number of consistency checks that ran.
Not started	Number of consistency checks that have not yet started in the current consistency checks run. Click the **Refresh** button to update this data during a currently executing check run.
In progress	Number of actively executing consistency checks at any given moment.
Finished	Number of consistency checks that completed in this run.
Start time	Time at which this set of consistency checks started.
End time	Time at which this set of consistency checks ended.
Duration	Length of time that this set of consistency checks took to run.
Version	Database version, listing (in order): • Application major version • Application minor version • Platform major version • Platform minor version • Data model version See "Understanding Guidewire software versioning" on page 400.
ID	Identifier (ID) of the stored results of this consistency check run.

If the consistency checks results table lists multiple check runs, use the check box next to a table row to select a consistency check run for further action.

See also

- "The View consistency checks definitions tab" on page 366
- "Run a consistency check from PolicyCenter" on page 367
- "Run a consistency check using system tools" on page 368
- "Correct a consistency check SQL failure" on page 368

The View consistency checks definitions tab

The **View consistency checks definitions** tab on the Server Tools **Consistency Checks Info** screen provides information on the consistency checks available for each database table. In this tab, you can undertake the following actions.

Action	Description
Download consistency check information	Click **Download** to download a ZIP file that contains a set of linked HTML files that describe the consistency checks that Guidewire provides in the PolicyCenter base configuration.
Search by table name	Search by table name to find the consistency checks related to a specified table. Most consistency checks operate on the specified table, but some checks, such as typelist table checks, operate on other tables as well. To search, enter a complete or partial table name in the **Table name fragment** field and click **Search**. The results of the search show in a table that lists the table name, the consistency check name, and a description of the consistency check. To clear the results of the search, click **Reset**.
Filter by check type	Filter the list of consistency check types to see a list of all tables for which that consistency check is available.
View SQL query	Review the SQL query used to generate a given database consistency check. First, select a consistency check, then: • Select the **Command** tab at the bottom of the screen 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 at the bottom of the screen (if available) 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.

See also

- "The Run Consistency Checks tab" on page 365
- "Run a consistency check from PolicyCenter" on page 367
- "Run a consistency check using system tools" on page 368
- "Correct a consistency check SQL failure" on page 368

Run a consistency check from PolicyCenter

Procedure

1. Navigate to the Server Tools **Consistency Checks** screen.
2. Select the **Run Consistency Checks** tab.
3. (Optional) Specify any, or all, of the following items:
   - Description that PolicyCenter prepends to the standard description of the tables and checks in the consistency check reports.
   - Tables or table groups on which to run the consistency checks.
   - Number of workers to use in executing the consistency check.
   - Type of consistency check to run.
4. Click **Run Consistency Checks**.
5. After the batch process completes, select one of the following in the summary table:

Download arrow	Downloads a Zip file that contains the full set of consistency check reports.
Download Errors arrow	Downloads a Zip file that contains only the consistency checks that contain SQL errors.
View icon	Opens a pop-up window from which you can view the full set of consistency check reports.

6. If you downloaded the consistency check file to your local system, unzip the file into its own directory.
7. Locate the `index.html` file and double-click it to open it in a browser.
8. In file `index.html`, 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.

See also

- "The Run Consistency Checks tab" on page 365
- "The View consistency checks definitions tab" on page 366
- "Run a consistency check using system tools" on page 368
- *Installation Guide*

Run a consistency check using system tools

Procedure

1. Open a command prompt.
2. Navigate to the following location in the PolicyCenter installation:

   ```
   admin/bin
   ```

3. Run the system tools command with the `-checkdbconsistency` option:

   ```
   system_tools -password password -checkdbconsistency [...]
   ```

   You must supply a value for *`password`*. You can optionally supply additional parameters for the `-checkdbconsistency` option.

See also

- "system_tools command" on page 429
- "The Run Consistency Checks tab" on page 365
- "The View consistency checks definitions tab" on page 366
- "Run a consistency check from PolicyCenter" on page 367

Correct a consistency check SQL failure

Before you begin

If a database consistency check fails due to a SQL error, PolicyCenter adds the following icons to the Server Tools **Consistency Checks** screen:

- **Download Errors**
- **Rerun**

Procedure

1. In the PolicyCenter Server Tools **Consistency Checks** screen, click **Download Errors** next to the check that generated the error.
2. Open the error report:
   a. Click the number in the **With Errors** column.
   b. On the details report that opens, click **Details**.
   c. Review the SQL query that generated the error.
   d. Correct any identified errors.

      The listed table name indicates the object against which the consistency check ran. The check name indicates which consistency check actually generated the error.
3. In the PolicyCenter Server Tools **Consistency Checks** screen, click **Rerun** next to the consistency check that generated the SQL error.

See also

- “The Run Consistency Checks tab” on page 365
- “The View consistency checks definitions tab” on page 366
- “Run a consistency check from PolicyCenter” on page 367
- “Run a consistency check using system tools” on page 368

Consistency check frequency

Guidewire recommends that you run all database consistency checks periodically during implementation and during the stabilization period before going into live production. Running the consistency checks during the implementation and stabilization phases frequently helps to discover issues with application configuration.

Upgrade

Guidewire recommends that you run all consistency checks before and after a database upgrade. Running these checks before and after a database upgrade helps to verify the validity of the data and identify potential issues.

See also

- “Run a consistency check from PolicyCenter” on page 367
- “Run a consistency check using system tools” on page 368

The Database Table Info screen

The Server Tools **Database Table Info** screen provides a way to access information about the tables used to store the PolicyCenter data model. The screen contains the following buttons:

Verify	Click to have PolicyCenter compare the database schema with the schema defined in the data model metadata files. PolicyCenter shows any errors that it finds on the screen.
Download Database Schema Verification Errors	Click to download the results of the database schema verification. If there were no errors, the report is empty.
Download Database Table Info	Click to download a ZIP file containing a number reports that document each table in the PolicyCenter data model.

See also

- “Database Table Info reports” on page 369
- “Understanding the Database Table Info reports” on page 370
- “View the Database Table Info reports” on page 370

Database Table Info reports

The **Server Tools→Info Pages→Database Table Info** screen provides the following reports.

Screen	Description
All Tables	Provides information about all PolicyCenter tables.
Guidewire Version	Lists schema version and build information for PolicyCenter.
Indexes by Table	Lists the indexes on a table and provides information about the associated key columns.
Spatial Indexes	Provides information about the spatial indexes.
Primary Key Constraints by Table	Lists the primary key constraints on tables and provides information about the fields that reference the keys.

Screen	Description
Foreign Key Constraints by Table	Lists the foreign key constraints on tables and provides information about the tables referenced by the keys.
Typekey Columns by Typelist	Lists the referencing typekey columns for each typelist.
Number of columns and min/max row lengths	Displays the number of columns and categories 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.
Possibly Redundant Backing FK Indexes	Lists foreign key indexes that may be redundant, including information about whether the index is unique and if it is an extension.
Indexes with Shared Prefixes	Lists indexes that share multiple leading key columns. It is possible to use this information to find redundant indexes.
Indexes with the Same Key Columns	Lists indexes that have the same key columns.
Indexes without a Description	Lists indexes that do not have a description.
Indexed Views	Lists any indexed views and the view definitions.
Event Paths from Tables to Listening Objects	Shows paths from event-generating entities to non-event-generating entities. Each row in the table contains a non-event-generating entity *E*, along with one of the paths from an event-generating entity to *E*. PolicyCenter uses each path to generate a query to find the event-generating entity instances that reference an instance of the non-event-generating entity.
Event Paths to Listening Tables	Shows the same paths as those in the **Event Paths from Tables to Listening Objects** report, but the entities in the second column are the event-generating entities. Each entry in the table contains an event-generating entity *E*, along with a path from *E* to a non-event-generating entity.
Instrumentation Queries	Lists the queries that PolicyCenter executes against the database while building the data the comprises the download reports.

PolicyCenter copies table-specific information from each report category to the individual report for the respective table, excepting the **All Tables** and **Instrumentation Queries** reports.

Understanding the Database Table Info reports

The report information that you download from the **Server Tools Info Pages→Database Table Info** screen consists of the following categories of information:

- Summary reports with links to detailed table reports
- Copies of specific configuration files.

Configuration files

The download ZIP file contains several directories of configuration files:

- The `config` directory contains a number of PolicyCenter configuration files as defined at server startup.
- The `current` directory contains the in-memory state of the `batch-process-config.xml`, `config.xml`, and `work-queue.xml` parameters on the server. Guidewire makes the in-memory state available because it is possible to change certain configuration parameters using a web service or JMX APIs after server startup.

View the Database Table Info reports

Procedure

1. Log into Guidewire PolicyCenter using an administrative account.
2. Navigate to the Server Tools **Info Pages Database Table Info** screen.
3. Click **Download Database Table Info**.

4. Save the download ZIP file.
5. Unzip the ZIP to a local directory.
6. Find and double-click file `index.html` to view a table of contents for the downloaded reports in a browser window.
7. Do one of the following:
   - Click a report type to open that report.
   - Click `config_files` on the table of contents page to access copies of certain metadata configuration files.

The Database Parameters screen

The Server Tools **Database Parameters** screen displays information about the database configuration. You can view the information on-screen, or you can download a set of linked HTML reports that contain the same information.

To download and access the HTML reports, click **Download Database Parameters Info**, unzip the resulting file, and click `index.html`.

To view the information on-screen, select a view type from the **View** drop-down list. Depending on the database type, the **View** list contains the following items.

View	Lists
Database and Driver	Versions of the database and its associated driver.
Database Connection Pool Settings	Connection pool settings as configured in `database-config.xml` if using PolicyCenter to manage the connection pool. See "The <dbcp-connection-pool> database configuration element" on page 242 for more information on these parameters. If you use the application server to manage the connection pool, then this screen does not show connection pool parameters. Instead, tune the connection pool by using the Administrative Console of the application server.
Database Connection Properties	Properties related to the database connection.
Guidewire Database Config	Guidewire-specific database configuration parameters. The table lists the `<database>` element attributes specified in file `database-config.xml` or lists the default value if file `database-config.xml` does not specify a value.
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 as defined by the `<upgrade>` element in file `database-config.xml`.
Guidewire Version	Information about this specific version of PolicyCenter.
Linguistic Search Options	Options defined for linguistic searching in PolicyCenter. See the *Globalization Guide* for more information.
Linguistic Search Oracle Functions and Java Source	Oracle functions, and Java source, for linguistic searching.
Oracle DB Options and Features	Oracle database options.
Oracle DBA Auto Tasks	Automated tasks defined for the Oracle database.
Oracle DBA Scheduler Jobs	Scheduled jobs for the Oracle database.
Oracle NLS Instance Params	Oracle database NLS (National Language Support) options.
Oracle NLS Session Params	Oracle database NLS (National Language Support) options.
Oracle Patch History	Patches applied to this Oracle database.
Oracle Permanent NLS DB Params	Oracle database NLS (National Language Support) options.
Oracle Registry	Oracle registry values.

View	Lists
Oracle Session Init Params	Initial parameters of the Oracle session for the current connection with PolicyCenter.
Oracle SGA Summary Info	Information about the Oracle System Global Area (SGA) shared memory.
Oracle State of Current Instance	Information on the current Oracle instance connected to PolicyCenter.
Oracle System Statistics	Information on Oracle database statistics.
Queries Executed to Build Download	SQL queries used to generate the information in the HTML download reports.
SQL Server Database Options	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 Global Server Settings	Global server settings for the SQL Server instance.
SQL Server Server Instance Attributes and Values	Attributes and values for the SQL Server instance connected to PolicyCenter.
SQL Server Session Properties	Properties of the SQL Server session for the current connection with PolicyCenter.
Summary of Queries Executed to Build Download	Number of queries used, and the execution time, to generate the information in the HTML download reports.

The Database Storage Information screen

The Server Tools **Database Storage Information** screen provides information about the space and memory taken up by the database on the PolicyCenter server. You can both view and download database storage information from this screen.

The following tables lists the filtering options for the database storage information:

Option	Available	Description
Include Estimation of Compression Savings for Tables and Indexes	Oracle SQL Server	If checked, you need to also select the compression level for estimating savings.
Select Compression Level for Estimating Savings	Oracle SQL Server	Only available if you chose to include estimation of compression savings in the database storage information. Options include: • Oracle – Advanced • SQL Server – Screen, Row
Collect Index Physical Stats for all tables	SQL Server	If you select **Yes**, then PolicyCenter includes statistics on all tables in the database. If you select **Specify tables**, PolicyCenter includes statistics only on the database tables that you select.
Select Mode for Collecting Index Physical Stats	SQL Server	Options include: • None • Detailed • Limited • Sampled Be aware that if you select the Detailed option without selecting individual tables the report can take a long time to generate.

After setting the desired filtering options, chose one of the following:

- To see the database storage information on the current PolicyCenter screen, click **Display Database Storage Info**.
- To download the database storage information to view later, click **Download Database Storage Info**.

After you click **Display Database Storage Info**, the screen shows information at the bottom of the screen, along with a drop-down that you can use to filter the information.

Oracle database options

To filter the information in the Server Tools **Database Storage Information** screen, select one of the following options from the **Storage Set to Display** drop-down filter:

Storage Set to Display	Description
Guidewire Version	Guidewire product-specific information such as application and platform version.
Indexes Alloc Space	Space allocation of each index on a table, in megabytes.
Oracle LOBs Alloc Space	Space allocation information for Large Object Blocks (LOBs), sorted by LOB name.
Oracle User LOBs	Space allocation information similar to that shown for **Oracle LOBs Alloc Space**, sorted by table name.
Queries Executed to Build Download	List of SQL queries used to generate the data. The data includes the SQL used to generate the query and other information such as the number of rows returned the time it took for the query to run.
Summary of Queries Executed to Build Download	Simple summary of the number of queries involved in generating the data and the total database time that the queries took to execute.
Table Alloc Space + Estimated Advanced Compression Settings	Size in megabytes for each table in the database. The information that you see depends on the filter options that you set: • If you select the **Include Estimation of Compression Savings for Tables and Indexes** option, PolicyCenter modifies the storage set name to indicate that fact and provides the requested information. • If you do not select the **Include Estimation of Compression Savings for Tables and Indexes** option, you see only **Table Alloc Space** as the storage set name. PolicyCenter does not show compression estimation savings information.
Tablespace Space	Size of each tablespace in megabytes, along with information on how much of the space is in use or is free space.
Tablespaces	List of each tablespace in the database along with related information.
User Indexes	Information on each user index in the database, selectable by index name.
User Tables	Information on each user table in the database, selectable by table name.

SQL Server database options

To filter the information in the Server Tools **Database Storage Information** screen, select one of the following options from the **Storage Set to Display** drop-down filter:

Storage Set to Display	Description
Data Spaces	Lists the filegroups taken up by the data and the amount of space taken and allocated by PolicyCenter.
Database Space	Details about the amount of disk spaced taken up by the database.
Guidewire Version	Guidewire product-specific information such as application and platform version. See “Understanding Guidewire software versioning” on page 400.
Index Physical Statistics + Estimated Database Compression Savings for Page Level Components	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** filter shows, select a new index.

Storage Set to Display	Description
	The information that you see depends on the filter options that you set: • If you select the **Include Estimation of Compression Savings for Tables and Indexes** option, PolicyCenter modifies the storage set name to indicate that fact and provides the requested information. • If you do not select the **Include Estimation of Compression Savings for Tables and Indexes** option, you see only **Index Physical Statistics** as the storage set name. PolicyCenter does not show compression estimation savings information.
Index Usage Stats	Information on the usage of an index on a table, selectable by table name.
Indexes with High Fragmentation	Display average percentage of fragmentation for indexes in the database.
Queries Executed to Build Download	List of SQL queries used to generate the data. The data includes the SQL used to generate the query and other information such as the number of rows returned the time it took for the query to run.
Summary of Queries Executed to Build Download	Simple summary of the number of queries involved in generating the data and the total database time that the queries took to execute.
Tables and Indexes	Lists paging and allocation type of a table and its indexes. To change which table the **Tables and Indexes** filter shows, select a new table.
TempDB Summary	Shows paging information for the database.

The Data Distribution screen

Use the Server Tools **Data Distribution** screen to a run batch processing job that generates data on the distribution of various items in the database. You can then view this information on-screen or download a set of reports that details this information.

There are multiple categories of data distribution reports.

Comparison reports	The report shows data for the various items selected for inclusion in any of the comparison reports. It also shows the individual row count for the data, as of that date. The intent of this report is to provide a way to visualize data growth. The download ZIP file contains an HTML report plus separate CSV reports. The HTML report contains distinct columns representing the data from each report used for comparison.
Combined reports	The report combines information from multiple runs. The intent of the report is to provide a way to create smaller reports that require less generation time and then combine the information into one report. The download ZIP file contains an HTML report that contains information on each of the previous reports combined into this report plus tables that contain the combined data.

It is also possible to start the data distribution batch process (`DataDistribution`) directly from the command prompt by using a `maintenance_tools` command option.

- "Generate and view a data distribution report" on page 374
- "Download comparison and combined data distribution reports" on page 375
- "maintenance_tools command" on page 426

Generate and view a data distribution report

Procedure

1. Navigate to the Server Tools **Info Pages** and select **Data Distribution**.
2. Select from the available options listed under **Data Distribution Batch Job Parameters**.
3. Click **Submit Data Distribution Batch Job**.
4. After the batch job completes, select one of the following in the summary table:

Download arrow	Downloads an `DataDistribution.zip` file that contains the set of database reports.
View icon	Open a pop-up window from which you can view the same reports contained in the `DataDistribution.zip` file, after you supply your user credentials.

5. If downloaded the report to your local system, unzip the download Zip file into its own directory.
6. Locate the `index.html` file and double-click it to open it in a browser.
7. Use the links on the screen to navigate through the distribution reports.

See also

- "The Data Distribution screen" on page 374
- "Download comparison and combined data distribution reports" on page 375

Download comparison and combined data distribution reports

Before you begin

To create either a comparison or a combined data distribution report, two or more data distribution reports and their output must exist in the database.

Procedure

1. Navigate to the Server Tools **Info Pages→Data Distribution** screen.
2. In the summary table, select (check) at least two of the generated reports.

   PolicyCenter enables the following download buttons:

   - **Download Comparison Zip File**
   - **Download Combined Zip File**
3. Click the appropriate button.

The Database Statistics screen

The Server Tools **Database Catalog Statistics Information** screen provides reports about out-of-date statistics in the database indexes, histograms, staging tables, and PolicyCenter application tables. This screen is not available with the development-only QuickStart database.

The **Database Statistics** screen contains the following tabs.

Tab	Description
Database Statistics Information	Use to generate and download database statistics reports for the entire database or for specific tables. See "Generate and download a database statistics report" on page 376.
Execution History	Use to view information about individual database statistics reports and take action with respect to each report. See "Working with database statistics reports" on page 376.
Oracle Statistics Preferences	Use to work with Oracle database preferences for database table statistics. This tab is only available after you set the `useoraclestatspreferences` attribute to `true` and perform an application upgrade. See "Using Oracle AutoTask for statistics generation" on page 292 for more information.

Database statistics generation

Batch processing type Database Statistics (`DBStats`) generates the data available on **Database Catalog Statistics Information** screen.

Development mode

In development mode, it is possible to run the Database Statistics proess in any of the following ways:

- From a command prompt, using the `-updatestatistics` option of the `system_tools` command
- From the **Execution History** tab of the Server Tools **Database Statistics** screen
- As a scheduled batch process

Production mode

In production mode, it is possible to run Database Statistics process in the following ways only:

- From a command prompt, using the `-updatestatistics` option of the `system_tools` command.
- As a scheduled batch process

Oracle AutoTask

For Oracle databases, it is possible to use the Oracle Autotask infrastructure to manage the collection of database table statistics. To use Oracle AutoTask, do the following:

- Disable any scheduled runs of `DBStats` batch processing.
- Set attribute `useoraclestatspreferences` attribute on the `<databasestatistics>` element in file `database-config.xml` to `true`.

You must use either Oracle AutoTask or `DBStats` batch processing to manage the collection of database statistics. Do not attempt to use both methods simultaneously.

See also

- "Understanding database statistics" on page 285
- "Managing database statistics using system tools" on page 287
- "Using Oracle AutoTask for statistics generation" on page 292
- "Generate and download a database statistics report" on page 376
- "Working with database statistics reports" on page 376
- "system_tools command" on page 429

Working with database statistics reports

To access the **Database Catalog Statistics Information** screens, click **Info Pages→Database Statistics** in the left-hand navigation pane of the **Server Tools** screen. Depending on the database type, you can access the following tabs from this screen:

Tab	For more information
Database Statistics Info	"Generate and download a database statistics report" on page 376
Execution History	"The Execution History tab" on page 377
Oracle Statistics Preferences	"The Oracle Statistics Preferences tab" on page 377

Generate and download a database statistics report

About this task

Use the Server Tools **Database Statistics** screen to generate database statistics about how the PolicyCenter application and data model interact with the physical database.

Procedure

1. Log into Guidewire PolicyCenter using an administrative account.
2. Navigate to the Server Tools **Info Pages→Database Catalog Statistics Information** screen.

3. Select one of the following values for the **View database catalogs statistics on all tables** option:

Yes	View statistics data for all database tables. This selection can take a few seconds to complete, depending on the size of your database.
No	View statistics data for the selected tables only. If you select this option and do not select any tables, PolicyCenter shows statistics data from the database metadata only.

4. Click **Download**.
5. Save the report ZIP file to a local directory.
6. Unzip the file.
7. Double-click file `index.html` to open the HTML report summary.
8. Use the report links to navigate through the various database statistics reports.

See also

- "Generate and download a database statistics report" on page 376
- "Working with database statistics reports" on page 376

The Execution History tab

The **Execution History** tab of the Server Tools **Database Statistics** screen lists information about each run of the batch process that updates database statics. This information includes the following:

- Date and time of the start and stop of the database statistics generation
- Type of database statistics generation, either full or incremental
- Description of the particular database statistics generation

From this screen, you can also perform the following actions.

Action	Description
Refresh	Update the contents of the summary table.
Run Incremental Statistics	Generate incremental database statistics. This action updates database statistics for tables exceeding the change threshold only. This functionality is available in development mode only.
Run Full Statistics	Generate full database statistics. This functionality is available in development mode only.
Download	Click the download arrow to download the data from this report.
Delete	Click the trash can icon to delete this data row from the summary table.

The Oracle Statistics Preferences tab

The **Oracle Statistics Preferences** tab presents information about the table statistics preferences set for the Oracle database. To view this information, navigate to the following location in Guidewire PolicyCenter:

Server Tools **Info Pages→Database Statistics** screen, **Oracle Statistics Preferences** tab

To be able to view the **Oracle Statistics Preferences** tab:

- The database is Oracle.
- Attribute `useoraclestatspreferences` on the `databasestatistics` element in file `database-config.xml` is set to `true`.

  Note: You must perform an upgrade (either full or rolling) after setting attribute `useoraclestatspreferences` to `true` in order for the change to the `useoraclestatspreferences` attribute to become effective.

Actual and configured table preferences

To configure database statistics in BillingCenter, one sets statistics options in file `database-configl.xml`, either at the global database level, or, at the individual table level. To be clear, you set statistics options in `database-configl.xml`, which BillingCenter then uses to set the actual table statistics preferences during the upgrade.

It is also possible to set table statistics preferences directly in the Oracle database by executing the following command:

```
DBMS_STATS.SET_TABLE_PREFS
```

If you set table statistics preferences directly, the actual table statistic preferences no longer match the configured table statistics preferences. During a database upgrade, BillingCenter ignores any actual table statistics preferences and sets table statistics preferences to those defined in the new `database-config.xml` file. The information provided on the **Oracle Statistics Preferences** tab provides a mechanism to capture direct changes made to the table statistics preferences that are lost during a database upgrade

Button actions

The following list describes the actions of the individual button in the **Oracle Statistics Preferences** tab.

Button	Action
Refresh	Refreshes the table information on the screen.
Download	Downloads the `database-config` table statistics options as a JSON file.
Reapply Config	Reapplies the table statistics options configured in file `database-config.xml` to the database, discarding any preferences set outside the application. As you initiate this process, BillingCenter inserts an entry into the application log and then inserts another entry at the end of the process.
Generate Config to Match Actual	Generates an XML file that details the table statistics preferences actually in use in the database.
Dev Mode only	(Drop-down) Selects that environment in which to generate the table statistics.

You can also filter the list of tables using the drop-down at the right of the tab.

The Oracle Statspack screen

The Server Tools **Oracle Statspack** screen provides a means to download HTML reports based on any two Oracle database statspack snapshots from the same instance start-up time. You create statspack snapshots by using a tool such as SQL*Plus or SQL Developer. Also, Oracle provides a script called `spauto.sql` that you can modify and run to automate statspack snapshot gathering.

The Server Tools **Oracle Statspack** screen is available only if the database server is Oracle. For PolicyCenter to display statspack information, you must also:

- Install the statspack package in your Oracle database (`spcreate.sql`).
- Grant `SELECT` privileges on all the `PERFTEST` tables to the PolicyCenter database user.

If you do not install the statspack or fail to grant the correct permission, PolicyCenter displays an error message on the screen. You cannot select any snapshots on the screen either. Refer to the Oracle documentation for statspack installation instructions.

Guidewire recommendations

Guidewire recommends the following with regards to Oracle statspack snapshots:

- Collect statspack snapshots at regular intervals if you do not have a license for Oracle AWR.
- Collect the snapshots at level 7 or higher to collect execution plan and segment statistics.
- Regularly purge statspack snapshots older than a certain period.

Guidewire recommends that you use the Guidewire AWR tool, rather than Oracle Statspack, if you have a license for the following:

- Oracle Diagnostics package
- Tuning package

If you do not have these licenses, Guidewire recommends that you acquire the two licenses.

See also

- “The Oracle AWR screen” on page 379

The Oracle AWR screen

The Server Tools **Oracle AWR Information** screen is available only if the database server is Oracle. Use the **Oracle AWR Information** screen to generate a set of Guidewire performance reports using AWR snapshots that you define in the database. The Guidewire AWR reports provide a view of the database activity that contains more detailed information than the Oracle Standard AWR report available with the Oracle database.

The Guidewire AWR reports provide the following additional information:

- Messaging analysis
- Concurrent batch processes
- Distributed worker activity
- Database statistics

The Guidewire AWR reports require that you have a license for the following:

- Oracle Diagnostics package
- Tuning package, if you select either **Probe in Memory SQL Monitoring** or **Probe on Disk SQL Monitoring**

Refer to the Oracle documentation for details.

See also

- “The Oracle Statspack screen” on page 378
- “Download an Oracle AWR unused indexes report” on page 381

Generate Guidewire AWR reports

About this task

Use the Server Tools **Oracle AWR Information** screen to generate a set of performance reports using AWR snapshots that you define in the database.

Procedure

1. Log into Guidewire PolicyCenter using an administrative account.
2. Navigate to the Server Tools **Info Pages→Oracle AWR Information** screen.
3. Set the options for the report.

   See “Automatic generation of Oracle standard AWR reports” on page 380 for a discussion of the **Include native Oracle report** option.
4. Select two database snapshots from the list at the bottom of the screen.

   Each snapshot must share the same Oracle instance startup time.
5. Click **Generate Perf Report**.
6. After PolicyCenter completes generating the report, select one of the following:

Download arrow	Downloads an `AWRReport.zip` file that contains the set of database reports.
View icon	Open a pop-up window from which you can view the same reports contained in the `AWRReport.zip` file, after you supply your user credentials.

Generate Guidewire AWR reports using system tools

Before you begin

Generating a Guidewire AWR report using system tools requires that you know the ID of two database snapshots.

Procedure

1. Open a command prompt.
2. Navigate to the following location in the PolicyCenter installation:

   ```
   admin/bin
   ```
3. Enter the following command to generate a list of database snapshot IDs:

   ```
   system_tools -password password -oraListSnaps numSnaps
   ```

   You must enter a value for *`password`*. You must limit the list of snapshots by entering a value for *`numSnaps`*.
4. Enter the following command to generate the Guidewire AWR report:

   ```
   system_tools -password password -oraPerfReport beginSnapshotID endSnapshotID
   ```

   The PolicyCenter administration command prompt tools all require that you enter the password of an administrative user for the tool to work. The use of a user name is optional.

   You must enter the IDs of two database snapshots.

Result

The `system_tools -oraPerfReport` command option reports the process ID of the process generating the performance report. You can check on the status of this process using the `-processstatus` option of the `maintenance_tools` command.

See also

- "system_tools command" on page 429

Generate Oracle standard AWR reports using a SQL script

About this task

Guidewire provides a SQL script that you can execute against the database server to generate the Oracle Standard AWR reports.

Procedure

1. Log into Guidewire PolicyCenter using an administrative account.
2. Navigate to the Server Tools **Info Pages→Oracle AWR Information** screen.
3. Generate and download an `AWRReport.zip` file as described in "Generate Guidewire AWR reports" on page 379.
4. Extract the contents of the `AWRReport.zip` file to a local directory.
5. Navigate to the following directory:

   AWRInfo→sqlscripts
6. Execute the following script against the database server:

   ```
   oraclescripts.sql
   ```

   Executing this script creates an Oracle Standard AWR report.

Automatic generation of Oracle standard AWR reports

Guidewire provides the means to generate an Oracle Standard AWR report automatically as you generate a Guidewire AWR report. To do so, ensure that you include report option **Include native Oracle AWR report** in the list of options against which to run the report. The PolicyCenter base configuration sets (checks) this option by default.

To generate a report with this option, the Oracle database user must have `EXECUTE` privilege for database package `DBMS_WORKLOAD_REPOSITORY`. If the database user does not have the required privilege, PolicyCenter does not generate a report and prints a message to the application log.

If the required privilege does not exist for the database user, do one of the following:

- Request that the database administrator grant the required privilege and rerun the report from **Oracle AWR** screen.
- Manually execute script `oraclescripts.sql` from the database server.

See also

- "Generate Oracle standard AWR reports using a SQL script" on page 380

The Oracle AWR Unused Indexes Information screen

The Server Tools **Oracle AWR Unused Indexes Information** screen provides the means to generate and download a report that contains information on the following types of indexes:

- Indexes that have no logical or physical reads
- Indexes that are not found in query plans

The **Oracle AWR Unused Indexes Information** screen is available only if the database server is Oracle.

See also

- "The Oracle AWR screen" on page 379

Download an Oracle AWR unused indexes report

About this task

The Server Tools **Oracle AWR Unused Indexes Information** report contains information on indexes that have no logical or physical reads or indexes that are not found in query plans.

Procedure

1. Log into Guidewire PolicyCenter using an administrative command.
2. Navigate to the Server Tools **Info Pages→Oracle AWR Unused Indexes Information** screen.
3. Set the options for the report.
4. Select two database snapshots from the list at the bottom of the screen.
   Guidewire recommends that you select snapshots that have a wide range.
5. Click **Download**.
6. In the download dialog, set the download location.
7. Click **OK** to download and save the report.
8. Open the downloaded ZIP file and click `index.html` to open the index to the linked set of report files.

The Oracle Outlines screen

The Server Tools **Oracle Outlines** screen is only available if the database server is Oracle. The screen contains information on any Oracle outlines defined in the PolicyCenter Oracle database. Oracle defines an outline as a collection of hints associated with a specific SQL statement, used to provide SQL execution plan stability. Consult the Oracle documentation on how to write and use Oracle Outlines.

The **Oracle Outlines** summary table contains the following information:

Name	Name of the Oracle outline. Click to open the **Outline Details** screen. This screen contains the SQL hints used to define the outline.
Category	Optional name used to group stored outlines.
Used	Whether the Oracle database has ever used this particular outline.
Time Stamp	Date and time of the last use of this outline.
Version	Oracle database version.

SQL	SQL query text that creates the database outline.
Signature	Unique identifier for this outline.
Compatible	Whether the stored outline is compatible with this database version.
Enabled	Whether Oracle enables this outline in the database.

Note: Oracle AWR contains a column that indicates if PolicyCenter used an Oracle Outline for a given SQL statement.

See also

- "The Oracle AWR screen" on page 379
- "View an Oracle Outlines report" on page 382

View an Oracle Outlines report

About this task

The Server tools **Oracle Outlines** screen contains information on any Oracle Outlines defined in the Oracle database.

Procedure

1. Log into Guidewire PolicyCenter using an administrative command.
2. Navigate to the Server Tools **Info Pages→Oracle Outlines** screen.
3. Click **Download**.
4. In the download dialog, set the download location.
5. Click **OK** to download and save the report.
6. Open the downloaded ZIP file and click `Outlines.html` to open the report.

See also

- "The Oracle Outlines screen" on page 381

The SQL Server Performance Report screen

The Server Tools **SQL Server Performance Report** screen is available only if the database server is SQL Server. From this screen you can do the following:

- Generate new SQL Server performance reports based on different parameters.
- View information about previously generated performance reports.
- View an existing performance report (development server only)
- Download an existing performance report.
- Delete an existing performance report.

The SQL Server performance report

SQL Server performance reports include the following types of data:

- SQL Server Dynamic Management Views
- SQL Server Query Store
- Database statistics

SQL Server Dynamic Management Views (DMV)

SQL Server provides Dynamic Management Views as a tool to monitor, diagnose, and tune database performance. Each performance report contains DMV information. PolicyCenter automatically generates this type of data and provides the data in each report.

SQL Server Query Store

SQL Server Query Store provides a set of tables separate from the Dynamic Management Views. Query Store persists data to disk, whereas DMV data is in-memory only and lost after a server restart. Thus, it is possible to obtain Query Store database statistics across server restarts.

To include Query Store statistics in the SQL Server performance report, select the **Include Query Store Analysis** option as you generate the performance report. If you select this option, then must also select the time interval the performance report as well.

For more information on SQL Server Query Store, refer the following web sites:

`Monitoring performance by using the Query Store`

`The SQL Server 2016 Query Store: Overview and Architecture`

Database statistics

The performance report can also include the same information as available from the Server Tools **Database Statistics** report. To include this information in the performance report, select the **Include database statistics** option as you generate the performance report.

Generating a SQL Server Performance report

It is possible to generate a SQL Server performance report using the following methods:

Generate a SQL Server performance report	More information
(Server Tools) **SQL Server Performance Report** screen	"Generate SQL Server performance report from PolicyCenter" on page 383
`system_tools` command option	"Generate SQL Server performance report using system_tools" on page 384

Enabling Query Store

It is possible to enable the use of Query Store in Guidewire PolicyCenter through the following methods.

Start Query Store from SQL Server Management Studio

Navigate to the **Query Store** screen in the SQL Server Management Studio and change the **Operation Mode (Requested)** option from **Off** to **Read Write**.

Start Query Store using Transaction-SQL

```
ALTER DATABASE database_name SET QUERY_STORE = ON
```

In addition, Guidewire automatically enables Query Store on SQL Server by default if you drop the PolicyCenter database using the `gwb dropDb` build command.

Generate SQL Server performance report from PolicyCenter

Generate a SQL Server performance report from the Server Tools **SQL Server Performance Report** screen.

Procedure

1. Log into Guidewire PolicyCenter using an administrative account.
2. In the left-hand navigation pane, navigate to following location in Server Tools.

   Info Pages→SQL Server Performance Report
3. Select the options to use in creating the report:

Include Database Statistics	Check to include database statistics similar to those on the Server Tools **Database Statistics** screen.
Include Query Store Analysis	Check to include analysis based on the SQL Server Query Store tables.

4. If you select the **Include Query Store Analysis** option, you need to select one to two time intervals from the table as well.

For the chosen time intervals:

- If you select a single time interval, the generated report includes data for that time period only.
- If you select two time intervals, the generated report includes data for the range of time defined by the two time intervals. After you generate the report, **SQL Server Performance Report** screen indicates the time range involved in the report description.

5. Click **Generate Perf Report**.

   You can view the progress of the report generation in the report table. Click **Refresh report list** to update the table information

6. After the batch process completes, select one of the following options in the summary table:

Download icon	Download a `DMVReport.zip` file that contains the set of database reports.
View icon	Open a pop-up window from which you can view the same reports contained in the `DMVReport.zip` file. This option is available on servers that are in development mode only.

7. After you download the `DMVReport.zip` file, unzip the file into its own directory.
8. Locate file `index.html` and double-click it to open it in a browser.
9. Use the links on the screen to navigate through the different types of report information.

Generate SQL Server performance report using system_tools

Use a `system_tools` command option to generate a SQL Server performance report.

Before you begin

Ensure that there is a running PolicyCenter server before starting this procedure and that you can access it as an administrative user.

Procedure

1. Open a command prompt on the running PolicyCenter server.
2. Navigate to the following location in your PolicyCenter installation:

   ```
   admin/bin
   ```

3. Enter the following command at the command prompt:

   ```
   system_tools -password password -mssqlPerfRpt firstID secondID collectstatistics
   ```

   The option parameters have the following meanings.

`password`	Required password for an administrative user.
`firstID`, *`secondID`*	Setting a single ID indicates a single time period. Setting two interval IDs indicates a time range. If you do not supply an interval ID, PolicyCenter does not generate Query Store statistics.
`collectstatistics`	Boolean value that indicates whether to include database statistics (`true`) in the report.

   For a discussion of the option parameters, see "About SQL Server reports and system tools" on page 384.

Next steps

To view the generated report, do one of the following:

- Download the report in the Server Tools **SQL Server Performance Report** screen.
- View the report in the Server Tools **SQL Server Performance Report** screen (development mode only).
- Download the report for viewing using the `-getPerfReport` option on the `system_tools` command.

About SQL Server reports and system tools

The `system_tools` command has a number of options that relate to SQL Server performance reports. Using these command options, you can do the following:

Action	Command option
Determine the IDs of the SQL Server performance report time intervals	`-sqlListIntervals`
Generate a performance report using the interval IDs	`-mssqlPerfRpt`
Determine the ID of each generated report	`-listPerfReports`
Download an existing report to a local directory using its ID	`-getPerfReport`

Note: All PolicyCenter command prompt tools require that you provide the password of an administrative user for the server on which you run the command.

`system_tools -password` *`password`* `-sqlListIntervals` *`n`*

Use this command option to list the *n* number of most recent Query Store runtime statistics intervals, including its ID. Each interval ID specifies a unique time interval. For example, with *n* is 3, the `-sqlListIntervals` option generates something similar to the following output.

```
3 most recent Query Store runtime stats intervals
ID=14, StartIntervalTime=Fri Aug 24 12:00:00 MDT 2018, EndIntervalTime=Fri Aug 24 12:00:00 MDT 2018
ID=13, StartIntervalTime=Fri Aug 24 11:00:00 MDT 2018, EndIntervalTime=Fri Aug 24 11:00:00 MDT 2018
ID=12, StartIntervalTime=Fri Aug 24 10:00:00 MDT 2018, EndIntervalTime=Fri Aug 24 10:00:00 MDT 2018
```

`system_tools -password` *`password`* `-mssqlPerfRpt` *`n1 n2 collectstatistics`*

Use this command option to generate a SQL Server performance report defined by the option parameters. If you do not know the interval ID, run the `-sqlListIntervals` command option. The `-mssqlPerfRpt` option parameters have the following meanings:

n1, *n2*	Optional interval IDs that indicate the time period to use for the SQL Server Query Store report: • Enter a single interval ID to generate a report for a single defined time interval. • Enter two interval IDs to generate a report across a time range. After PolicyCenter generates the report, the **SQL Server Performance Report** screen indicates the time period used for the performance report.
collectstatistics	A Boolean value that determines whether PolicyCenter includes database statistics (`true`) in the output report.

Depending on how you set the command options, this command option generates something similar to the following output.

```
Submitting SQL Server Performance Report from interval 15 to 16 with collect statistics set to true
View results on the SQL Server Performance Report Page or download it here.
```

To view the resulting report, do one of the following:

- Navigate to the Server Tools **SQL Server Performance Report** screen.
- Download the report to your local drive by running the `-getPerfReport` option.

`system_tools -password` *`password`* `-listPerfReports` *`n`*

Use this command to list the *n* most recent SQL Server performance reports, including the report ID. For example, if *n* is 2, the `-listPerfReports` command option generates something similar to the following output.

```
Most recent database performance downloads
ID: pc:5, Status: Succeeded, Start: 08/24/2018 1:29 PM, End: 08/24/2018 1:37 PM, Description: Report
      for interval 15, not including database statistics
ID: pc:4, Status: Succeeded, Start: 08/24/2018 1:11 PM, End: 08/24/2018 1:18 PM, Description: Report
      without Query Store, not including database statistics
```

`system_tools -password` *`password`* `-getPerfReport` *`ID`*

Use this command to download a copy of the SQL Server performance report specified by *`ID`* to a local drive. If you do not know the report ID, run the `-listPerfReports` option. This command option generates something similar to the following output.

```
Database performance report stored in
        C:\Guidewire\10.0\pc10\admin\bin\DbPerfReport_Fri_Aug_24_14.58.30_MDT_2018.zip
```

To download the report file to a directory other than the default directory, add the `-filepath` option to the `-getPerfReport` command option.

See also

- "system_tools command" on page 429

The Microsoft JDBC Driver Logging screen

The Server Tools **Microsoft JDBC Driver Logging** screen is available only if the database server is SQL Server. Use this screen to specify the following:

- The logging level for the Microsoft JDBC driver. Be cautious setting the logging level, as detailed logging can slow the system significantly.
- The logging format, either **Simple**, for a more readable format, or **XML** for XML output, usually parsed by another system.
- The log file location, adding special components that PolicyCenter replaces at runtime. For example, use `%u` to append a unique number to each log to avoid conflicts.

If you have already enabled logging through `database-config.xml` or from previous use of this screen, use this screen to make additional changes to reset the logging level. After clicking **Set Logging Level**, PolicyCenter flushes and closes any existing logging files before beginning the new trace. If you choose `OFF` as the logging level, PolicyCenter disables any current logging as well as flushing and closing any existing logging files.

The ability to control logging of the Microsoft JDBC driver through PolicyCenter only works if using the internal connection pool, not if using an external JNDI data source connection pool.

Note: Using this screen is a better option if tracing a particular operation, in order to minimize system impact and size of the trace file.

See also

- *Installation Guide*

The Load History Information screen

The Server Tools **Load History Information** screen displays information about specific PolicyCenter database operations. For example, loading data into the staging tables impacts the loader history information on this screen.

On this screen:

- Click **Refresh** to reload and update the table data.
- Click **Edit** to make the **Description** field for each load history writable.

The load history summary table contains the following information:

Download	Click the **Download** arrow to download a `LoadHistoryInfo.zip` file that contains a set of HTML reports. The reports consist of a summary table and a set of links to individual reports that load different views of the database operation data onto the screen. These are the same reports that are available by clicking the **View** icon.
View	Click the **View** icon to view the on-screen **Load History Detail** report for the selected database operation. The detail view consists of a summary table and a set of tabs that load different views of the database operation data onto the screen. These are the same reports that are available for download by clicking the **Download** icon.
Delete	Click the trash can icon to remove the data for this database operation.

Load Operation Type	Type of database operation. For example, this can be any of the following: • Database table load operations • Staging table clearing operations • Database statistics generation operations
Start Time	Start date and time of the database operation.
End time	Completion date and time of the database operation.
Duration	Length of time, in seconds, to complete the operation.
Error Count	Number of reported errors for this database operation..
Calling User	Name of user who initiated this database operation.
Description	Click **Edit** to make the **Description** field for each database operation row writable. Click **Update** after entering text to save your work and update the field.

See also

- "View a load history report" on page 387
- "The load history detail report" on page 387

View a load history report

Procedure

1. Log into Guidewire PolicyCenter using an administrative account.
2. Navigate to the Server Tools **Load History Information** screen.
3. For the database load operation that interests you, select one of the following in the summary table:

Download	Click the arrow icon to download a `LoadHistoryInfo.zip` file that contains the set of database reports.
View	Click the view icon to open a new screen that displays the **Load History Detail** report.

4. If you downloaded the report file, unzip the file into its own directory.
5. Locate the `index.html` file and double-click it to open it in a browser.
6. Use the links on the screen to navigate through the linked reports.

The load history detail report

The Server Tools **Load History Detail** report contains a summary table for the selected database load operation. As with the main **Load History Information** table, the report summary view lists information about the type of database operation, the user who initiated the operation, and similar information.

Underneath the report summary table is a set of tabs, each of which loads a different view of the load operation data onto the screen. The following list describes these tabs.

Parameters	Lists the values of the configuration parameters used in generating the data for the database operations reports. See "Load history detail report - Parameters tab" on page 388 for more information.
Steps	Lists the individual steps in the data load operation. Click a step to view detail data about that step. See "Load history detail report - Steps tab" on page 388 for more information.
Row Counts	Lists information about database tables impacted by the data load. Use the information on this screen to quickly assess whether the amount of data loaded by operation was the amount that you expected the operation to load.
Integrity Checks	Lists the integrity checks that PolicyCenter ran against each affected database table before the data load operation.

Inserts	Lists the results of the SQL `INSERT_INTO` queries that PolicyCenter ran against the affected database tables. This tab lists all `INSERT_INTO` run for this database load. In contrast, the **Steps** tab lists only the `INSERT_INTO` queries that run for each particular step.
Callbacks	Lists the operations that PolicyCenter executes before and after a staging table load operation. This tab lists all callback operations for this database load. In contrast, the **Steps** tab lists only the callback operations that execute for each particular step.
Statistics Commands	Lists the SQL commands used to generate database statistics.

Load history detail report - Parameters tab

The Server Tools **Load History Detail** report contains a summary table for the selected database operation. Directly underneath this table is a **Parameters** tab. Selecting this tab opens a table that lists the database parameters that affect the results of the **Load History Detail** report. The table shows the current values of these parameters.

Setting parameters for the load history report

The following table describes how to set the database parameter values that affect the results of the load history detail report.

Parameter	How to set
`allowRefsToExistingNonAdminRows`	Use the `table_import -allreferencesallowed` command prompt option.
`clearErrorTable`	Use the `table_import -clearerror` command prompt option.
`parallelism`	Modify file `database-config.xml`.
`populateExclusionTable`	Use the `table_import -populateexclusion` command prompt option.
`updateDBStatisticsWithEstimates`	Use the `table_import -estimateorastats` command prompt option.

Load history detail report - Steps tab

The Server Tools **Load History Detail** report contains a summary table for the selected database operation. Directly underneath this table is a **Steps** tab. Selecting this tab opens a table that lists the steps involved in the operation in this particular load operation.

Steps and Ops link

If a step contains multiple sub-steps, that step becomes a clickable link that opens a **Steps and Ops** detail screen for that step. The **Steps and Ops** detail screen contains a summary table that describes the substeps involved in this step operation. It can also contain an additional table depending on the type of step. For example, both callback and `INSERT_INTO` operations generate an additional table of database information.

Be aware that the **Table Name** column can contain the name of an actual database table or the name of a database callback. If it is a callback, then the associated **Callbacks** table contains specific information about that callback operation.

The Load Integrity Checks screen

The Server Tools **Load Integrity Checks** screen reports on the SQL integrity checks that run as a database load operation executes. For each integrity check, the screen lists the SQL query that the check performs, along with a description of the integrity check.

This screen has two tabs.

View by Staging Table	This tab lists the set of integrity checks that PolicyCenter executes against a specific staging table. Use the **Staging Table** drop-down to select the table of interest.

View by Load Error type	This tab lists the set of tables against which PolicyCenter runs a particular integrity check. Use the **Load Error Type** drop-down to select a specific integrity check.

In both tabs, you are able to set the value of **Allow Non Admin References** to true. If you do, PolicyCenter checks foreign key references to administrative tables on load, such as users and groups. In the base configuration, Guidewire disables these references by default.

See also

- *Integration Guide*

The Load Errors screen

The Server Tools **Load Errors** screen displays errors generated by failed integrity checks. You can use this screen 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** screen 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.

The Runtime Environment Info screen

The Server Tools **Runtime Environment Info** screen lists information about the runtime environment for PolicyCenter. This information includes Guidewire platform and PolicyCenter build information, system properties, and environment variables.

The Safe Persisting Order screen

The Server Tools **Safe Persisting Order** screen provides information on the following:

- The order in which PolicyCenter writes object data to the database during a bundle commit.
- The order in which PolicyCenter runs the Preupdate rules on a set of objects.

The screen contains a table with columns that show the following:

Column	Description
Entity	Entity name
Order	Entity ranking order
Table	Entity database table name
Preupdate	Whether the entity triggers a Preupdate rule set

To see only those objects affected by Preupdate rules, select **With rules only** from the drop-down filter.

Note: PolicyCenter does not use Preupdate rules.

Bundle commits to the database

PolicyCenter uses the defined object ordering in every bundle write to the database for insert, write, or delete operations. Thus, during a commit to the database of a bundle containing multiple objects, PolicyCenter writes the objects to the database in the order listed in the table **Order** column.

Preupdate rules

In running the rules in the Preupdate rule set, PolicyCenter first computes the set of objects on which to run the Preupdate rules. PolicyCenter then runs the Preupdate rules for this set of objects as determined by the order listed in the table **Order** column.

The Loaded Gosu Classes screen

Guidewire no longer supports the Server Tools **Loaded Gosu Classes** screen.

The Serialization Info screen

The Server Tools **Serialization Info** screen (under **Info Pages**) shows, for any specific server in the cluster, the entire set of Java objects (classes) deserialized by that server instance. This screen contains an optional filter (**Including listed in the serialization whitelist classes**) that filters the list of classes:

- Checking this box means that the list of class names includes the names of all Java classes encountered and deserialized by the local server. This list includes the names of classes that exist in the serialization white (permitted) list as well.
- Un-checking this box means that the list of class names includes only the names of classes encountered and deserialized that are not on the serialization white list. Guidewire recommends that you add these classes to the serialization whitelist. After you complete your whitelisting of Java classes, the class listing will be empty.

Enabling object deserialization

Configuration parameter `SerializationWhitelistEnabled` in `config.xml` determines whether PolicyCenter permits only those Java classes placed on a serialization whitelist to be deserialized. Before you enable the use of the whitelist, ensure that you first add any additional types needed due to customizations. Primarily, this is due to creating batch processes that accept custom objects as arguments. Such objects must be serialized if the batch process is invoked from a server other than the server with the batch role. If you do not add these objects to the whitelist, it is possible that the related services that uses them to not function properly.

See also

- *Configuration Guide*

The black and white serialization lists

In Guidewire Studio, you can access the serialization black list and white lists in the following location:

configuration→config→security

To blacklist a Java class, add an entry in `serialization-blacklist.lst` in that folder. To whitelist a Java class, add an entry in `serialization-whitelist.lst` in that folder.

In making entries in these files, use the following syntax:

- Place a `#` symbol at the beginning of a line to indicate that the line is a comment.
- Use a separate line for each class or package name, for example, `gw.api.myPackage.*`.
- Do not place the `*` separator in the middle of a class or package name. For example, do not do the following:

```
#Incorrect example
gw.api.*.myClass
```

- Use blank lines and leading spaces as desired to enhance readability of the file.

The Management Beans screen

Note: The **Management Beans** screen is accessible to users with the `soapadmin` permission only.

The Server Tools **Management Beans** screen lists PolicyCenter management beans, which are PolicyCenter objects that represent different resources. Click the name of a resource to open the **Guidewire Managed Bean Properties** screen for the selected resource.

The **Bean Properties** screen most often contains a **MBean Property** table that lists the properties associated with the selected resource bean. Property values are either read-only or editable. Guidewire marks the editable properties with a small blue triangle in the upper left-hand corner of the **Value** field. Click anywhere in an editable field to make that field editable. After modifying a field, you can either save your work or cancel your changes by clicking **Save** or **Cancel**.

A few **Bean Properties** screens also contain an **Operation** table that lists available operations associated with this resource bean. Click **Execute** to execute a selected operation. The **Result** field shows the result of the operation.

The Startable Services screen

The Server Tools **Startable Services** screen contains summary information on all of the startable plugin services in the PolicyCenter cluster. A startable plugin is a special type of PolicyCenter plugin. At runtime, PolicyCenter creates a background process, or service, for each startable plugin. The summary table provides the following information for each service.

Name	Name of the plugin service.
Status	Current status of the service.
Host	One of the following: • Name of the host on which the plugin service is running. • Distributed, if cluster configuration allows the startable plugin services to run on all cluster members.
Action	Available actions, either starting or stopping a selected service.

See also

- *Integration Guide*

The Cluster Members screen

The Server Tools **Cluster Members** screens provide information on the server cluster installation. The screen consists of the following areas:

- **This Application Instance**
- **Application Server Instances**
- **Components** tab
- **History** tab

From this screen you can also schedule (or cancel) a planned shutdown of a specific server in the PolicyCenter cluster.

See also

- "The Components screen" on page 395
- "Download a server component report" on page 397
- "Schedule a planned cluster member shutdown" on page 397

Cluster Members (This Application Server Instance)

The Server Tools **Cluster Members (This Application Server Instance)** screen provides the following information about the local server instance from which you access the screen.

Host	Name of the machine on which this PolicyCenter server instance is running.

Server ID	Name for this server instance. You specify the server ID by either adding an entry for this cluster member in the `<registry>` element in `config.xml`, or, by setting a JVM option at server startup. If you do not specify a server ID, PolicyCenter uses the host (machine) name as the server ID.
UUID	Universally Unique ID for this server machine. PolicyCenter randomly generates a UUID for each machine at each machine startup.
Server Roles	List of server roles assigned to this PolicyCenter server.

See also

- "Understanding the configuration <registry> element" on page 60
- "Server roles" on page 169

About planned server shutdowns

The Server Tools **Cluster Members** screen provides the ability to start a planned or scheduled shutdown of a specific server in the PolicyCenter cluster. This action affects only the server on which you start or stop the scheduled shutdown. The **Actions** column of the **Application Server Instances** table contains a button that toggles between **Start planned shutdown** and **Cancel planned shutdown**, depending on whether a server shutdown is currently scheduled.

Scheduling a server shutdown

Clicking **Start planned shutdown** opens the **Schedule Planned Shutdown** screen. From this screen, you can set the details of the planned server shutdown. In this screen, you need to set the following items:

Banner text	Choose the text of the banner message to show on the PolicyCenter screen to warn users of the server shutdown. The banner message occurs immediately as you schedule the shutdown and contains a countdown to the time of the scheduled shutdown. You can choose from several listed messages or create your own custom message.
Shutdown date and time	Set the date and time of the server shutdown.
Action to take with respect to running batch processes	Decide how to handle any currently running batch processes

After you initiate a scheduled shutdown, PolicyCenter does the following on the affected server:

- It does not start any new batch processes.
- It manages the stopping of any currently running batch processes depending on the setting for **Terminate Batch Processes** field.
- It requests that all work queues and message destinations stop processing immediately.
- It completes the transmission of any current messages without starting any new message transmissions.
- It completes the processing of any current work items without beginning work on any new work items.

After you click **OK** in the **Schedule Planned Shutdown** confirmation dialog, PolicyCenter reopens the **Cluster Members** screen. In this screen, you now see the following:

- The **Actions** entry for the affected server contains a **Cancel planned shutdown** button.
- The **Planned Shutdown** entry for the affected server provides information about the planned shutdown.
- The screen contains a banner with the chosen shutdown message and a countdown timer to the server shutdown date and time.

Shutdown message details

You have several choices for shutdown messages in the **Schedule Planned Shutdown** screen. If you do not elect to create a custom message, you can select one of several default messages. PolicyCenter stores these messages as display keys in file `display.properties`.

`Web.TabBar.SystemAlertBar.PlannedShutdown.RollingUpgradeMessage`	Rolling upgrade in progress, please save your work and log out to redirect to a new server.
`Web.TabBar.SystemAlertBar.PlannedShutdown.ScaleInMessage`	Please save your work and log out to re-direct to the new server.

After you initiate a planned shutdown, PolicyCenter stores only the current time and the shutdown time in the database. It stores other information, such as the shutdown message itself, in a single-threaded atomic reference in memory. PolicyCenter clears this message reference under the following conditions:

- At the restart of the shutdown PolicyCenter server
- At the cancellation of the planned server shutdown

Logging into PolicyCenter during a planned server shutdown

Guidewire does not restrict the ability of an credentialed user to log into a PolicyCenter server that is undergoing a planned shutdown. However, upon logging into the server, that user sees the alert banner that a planned shutdown is in progress with a message instructing the user to log out. If you want to redirect users away from the server undergoing shutdown, Guidewire recommends that you delegate this function in the load balancer configuration.

Terminating running batch processes

The **Schedule Planned Shutdown** screen contains a **Terminate Batch Processes** checkbox. PolicyCenter terminates the running batch processes differently depending how you set this field.

Terminate Batch Processes	Result
Checked	PolicyCenter sets a flag on all currently running batch processes (including workqueue writers) assigned to this server to terminate their operation. To the extent that the batch process logic permits, the process attempts to stop as gracefully and as safely as possibly.
Unchecked	PolicyCenter does not instruct the currently running batch processes on this server to terminate prematurely. All batch processes assigned to this server continue until completed, regardless of how long it takes to complete the work.

In either case, PolicyCenter does not show a **Planned Shutdown** status of ready on the **Cluster Members** screen until all the affected processes have stopped on the server.

> **Note:** Batch processes run as non-daemon threads. As such, it is not possible for the server to perform a graceful shutdown if any batch processes are still running on the server at the time of the shutdown.

Using system tools to schedule a shutdown

It is also possible to initiate a scheduled shutdown of a server using either the `SystemToolsAPI` web service or its associated `system_tools` command prompt option. For example, the `system_tools` command uses the following syntax:

```
system_tools -user user -password password -scheduleshutdown serverId [-
terminatebatchprocesses -shutdowndelay minutes]
```

If you include the `-terminatebatchprocess` command option, the shutdown process is the same as if you checked the **Terminate Batch Processes** check box in the **Schedule Planned Shutdown** screen.

For more information on the use of the `system_tools` command, see “system_tools command” on page 429. For information on the use of the `SystemToolsAPI` web service, see the PolicyCenter *Integration Guide*.

Cluster Members (Application Server instances)

The Server Tools **Cluster Members (Application Server Instances)** table provides the information about individual server instances recognized by the cluster. To update this information, click **Refresh Cluster Members**. A review of this information is one way to determine if a cluster member has become unreachable, and therefore, missing from the list.

This screen provides the following information.

Server ID	Name for each PolicyCenter server in the cluster.
Status	Status of each server instance in the cluster.
Host	Name of the machine on which each PolicyCenter server is running.
User Sessions	Number of user sessions active on each server instance in the cluster.
Run Level	Run level for each server instance in the cluster.
Version	Version information for each application installation. PolicyCenter provides the version information in the following format: Guidewire build.customer build (*n*,*n*,*n*,*n*,*n*) These numbers have the following meaning: • Guidewire build – Guidewire application version, for example, 10.0.0. • Customer build– Custom label, defined in file `customer-version.properties`. If not defined, this field is empty. • (*n*,*n*,*n*,*n*,*n*) – Numbers that have the following meaning: ◦ Platform major version ◦ Platform minor version ◦ Application major version ◦ Application minor version ◦ Data model version See “Understanding Guidewire software versioning” on page 400 for more information.
Server Roles	Specific roles assigned to each server instance in the cluster.
Server Started	Date and time at which this server instance started.
Connection Started	Date and time this cluster member first started.
Last Update	Date and time of the last update on this server instance.
Planned Shutdown	Date and time of any planned shutdown of a cluster member.
Actions	Action button to manage a planned shutdown of a cluster server instance. The button label is either **Start Planned Shutdown** or **Cancel Planned Shutdown**. You can also initiate a server shutdown using the administrative `system_toools` command option `-scheduleshutdown`. See “system_tools command” on page 429 for details.

See also

- “The Components screen” on page 395
- “Schedule a planned cluster member shutdown” on page 397

Cluster Members (Components tab)

The Server Tools **Cluster Members – Components** tab provides information on the components running on the server that you select in the **Cluster Members (Application Server Instances)** table. These components, with the exception of the work queues, are all singleton components. You can view all components, or, filter the list to see only those components of a specific type.

The **Components** table provides the following information about the components running on the selected server.

Type	Type of component, one of the following: • Batch Process • Message Destination • Startable Service • System • Work Queue
Name	Name of component. This name is the work queue name, or, the message destination name, for example.
Started	Date and start time for each component.
State	State of the component, for example, Assigned.
Retry Failover	Date and time of the deadline in which to complete the failover process. If a lease manager detects the expiration of a component lease owned by another cluster member, the first lease manager starts a failover process for this lease. This failover process is not instantaneous. It is possible that during the failover process the cluster member performing the failover itself might fail, or lose database connection, or encounter other potential problems. To be able to handle this situation gracefully, the cluster member starting the failover gives itself a deadline to complete the failover process. If the current failover process does not finish by the specified timestamp, some other cluster member needs to start the failover process for the same component lease. See "Automatic failover of a component lease" on page 188 for a description of how PolicyCenter calculates this timestamp.

You can view similar information to that of the **Components** table on the **Cluster Components** screen.

Cluster Members (History tab)

The Server Tools **Cluster Members – History** tab provides information on the history of the server that you select in the **Cluster Members (Application Server Instances)** table.

The **History** table provides the following information for the selected server.

Host	Name of the machine on which the PolicyCenter server is running.
UUID	Universally Unique ID for this server machine. PolicyCenter randomly generates a UUID for each machine at each machine startup.
Env	Environment for this PolicyCenter server.
Last Run Level	Run level for this PolicyCenter server at the point it stopped
Server Roles	Specific roles assigned to this server instance.
Server Started	Date and time at which this server started.
Server Stopped	Date and time at which this server stopped.

The Components screen

The Server Tools **Components** screen provides the following information.

Type	Type of component to show in the table, one of the following: • All types • Batch Process • Message Destination • Startable Service
Name	Name of component. This name is the work queue name, or, the message destination name, for example.
State	State of the component, for example, Assigned, meaning that there is a server that has the lease to run this component.

Start Requested	Date and time that the server received the component start request.
Started	Date and start time of the component.
Owner	Server ID of the PolicyCenter server on which the component runs.
Lease Expiration	Date and time at which the component lease expires. See "Cluster member shutdown" on page 173 for more information on component leasing.
Stopped	Stop date and time of the server instance on which the component is running, due to one of the following reasons: • Server instance stopped as intended • Server instance failed due to lost power or network connection and automatic recovery failed or is still in progress
Actions	PolicyCenter shows a **Complete Failover** button if a component lease expires and a custom failover plugin implementation instructs PolicyCenter not to perform an automatic failover. See "Cluster member shutdown" on page 173 for more information.
Terminate Requested	Date and time at which the server instance received a terminate request for this component.
Transfer Requested	Date and time at which the server instance received a request to transfer this component to another server instance.
Transfer Target	Server ID of the server to which the transfer request was made.
Retry Failover	Date and time of the deadline in which to complete the failover process. If a lease manager detects the expiration of a component lease owned by another cluster member, the first lease manager starts a failover process for this lease. This failover process is not instantaneous. It is possible that during the failover process the cluster member performing the failover itself might fail, or lose database connection, or encounter other potential problems. To be able to handle this situation gracefully, the cluster member starting the failover gives itself a deadline to complete the failover process. If the current failover process does not finish by the specified timestamp, some other cluster member needs to start the failover process for the same component lease. See "Automatic failover of a component lease" on page 188 for a description of how PolicyCenter calculates this timestamp.

You can view similar information to that of the **Components** table on the **Cluster Members** screen, the **Components** tab.

Available actions on this screen

The following list describes the actions that you can take in **Cluster Members** screen.

Action	Description
Download cluster server report	Click **Download** to download an HTML report of the cluster components and component history. See "Download a server component report" on page 397 for details.
Filter by component type	Select a component type from the **Types** drop-down list to filter the information by a specific component type.
Filter by component state	Select a component state from the **State** drop-down list to filter the information by a specific component state.
Filter by component	Click **Filter by Component** to open a **Select Components** screen. In this screen, you can select individual components of all available types to show in the component table.
Refresh the component information	Click **Refresh** to update the server component information in the table.
Review component history detail	Click the name of a component to open a component history detail screen.

Download a server component report

Procedure

1. Log into Guidewire PolicyCenter using an administrative account.
2. Navigate to the Server Tools **Cluster→Components** screen.
3. Click **Download** to open the **Specify Record Limits for this Report** screen.
4. Enter the maximum number of days to include in the generated component history report.
5. Click **Complete Download** to generate a ZIP file that contains the actual report.
6. Save the download file to your local machine.
7. Extract the Zip file to a local directory.
8. Search for and double-click file `index.html` to open the report in a web browser.
9. Click the **Component History** link for the component history report.

Schedule a planned cluster member shutdown

About this task

You schedule a planned shutdown of a cluster member from the Server Tools **Cluster Members** screen.

Note: It is also possible to initiate a planned server shutdown using the administrative `system_toools -scheduleshutdown` command option. See “system_tools command” on page 429 for details.

Procedure

1. Navigate to the Server Tools **Cluster Members** screen.
2. Find the table row that corresponds to the cluster member for which you want to schedule a shutdown.
3. If necessary, scroll the screen all the way to the right until you see the **Actions** column.
4. Click **Start Planned Shutdown**.
5. In the **Schedule Planned Shutdown** screen, select the message that you want to show to the application user about the planned shutdown.

   Select one of the provided messages, or, enter custom text in the provided field.
6. Select the date and time of the planned shutdown.
7. Decide how you want PolicyCenter to handle the currently running batch processes (including work queue writers) on the server.

   See “About planned server shutdowns” on page 392 for more information on the option.
8. Click **Schedule Shutdown**.

Result

After you schedule the shutdown, the **Cluster Members** screen shows the following for your selected cluster member:

- The **Planned Shutdown** column shows the date and time that you initiated the planned shutdown. It also shows the date and time of the actual planned shutdown.
- The **Actions** column shows a **Cancel Planned Shutdown** button. To cancel a planned server shutdown, click **Cancel Planned Shutdown**.

All PolicyCenter screens, for all users logged into the affected cluster member, display a banner indicating the date and time of the planned shutdown and your selected message.

The Upgrade and Versions screen

The Server Tools **Upgrade and Versions** screen displays information about any upgrade process that is run on this server.

Note: Some functionality on the **Upgrade and Versions** screens is visible only if configuration parameter `ClusteringEnabled` is set to `true` in `config.xml`.

Guidewire divides this screen into the following areas:

- A row of buttons that control upgrade functionality, not all of which are visible at all times.
- A summary table that contains upgrade information for this particular application server.

The upgrade and versions screen summary table

The upgrade summary table contains the following information.

Version	Version information for each application installation. PolicyCenter provides the version information in the following format: Guidewire build.customer build (*n*,*n*,*n*,*n*,*n*) These numbers have the following meaning: • Guidewire build – Guidewire application version, 10.0.0, for example. • Customer build– Custom label, defined in file `customer-version.properties`. If not defined, this field is empty. • (*n*,*n*,*n*,*n*,*n*) – Numbers that have the following meaning: ◦ Platform major version ◦ Platform minor version ◦ Application major version ◦ Application minor version ◦ Data model version See “Understanding Guidewire software versioning” on page 400 for more information.
Status	Status of each upgrade, for example, new schema or upgraded.
Type	Type of each upgrade, for example, install or full.
Start Time	Date and time of the beginning of the upgrade.
End Time	Date and time of the completion of the upgrade.
Duration	Length of time that the upgrade process took. Time is shown as both the total number of seconds involved and the time for just the database upgrade portion of the upgrade.
Deferred Upgrade Tasks Status	Execution status of the Deferred Upgrade Tasks batch process. See “Deferred Upgrade Tasks batch process” on page 139 for details.
View Details	Click the **View** icon to open a pop-up from which you can view the same reports contained in the `UpgradeInfo.zip` file. See “View an upgrade report” on page 399 for more information.
Download Details	Click the **Download** arrow to download an `UpgradeInfo.zip` file that contains a set of HTML reports describing various aspects of the upgrade process. See “View an upgrade report” on page 399 for more information.
Remove Detail Data	Click the trash can icon to remove this row of data.

The upgrade and versions screen buttons

The top of the **Upgrade and Versions** screen contains a number of buttons that initiate specific upgrade actions. Not all buttons are visible at all times. Individual buttons become visible depending on various aspects of the cluster and its members.

The **Upgrade and Versions** screen contains the following buttons that initiate upgrade-related actions.

Refresh	Click to update the information on this screen.

Start Rolling Upgrade	Click to open a **Start Rolling Upgrade** screen from which you can initiate a rolling upgrade for this cluster. Before you can start the upgrade, you must actively affirm that you have taken certain pre-upgrade steps by selecting the check box next to each pre-upgrade step. See "Performing a rolling upgrade" on page 204 for details.
Rolling Upgrade Complete	Click to notify PolicyCenter that the in-progress rolling upgrade is now complete. The rolling upgrade is complete after you have restarted all cluster members with the new target configuration. Without this notification, PolicyCenter prevents cluster members with the old configuration from starting up again.
Initiate Backout	Click to attempt to stop an in-progress rolling upgrade and initiate a backout of the new target configuration. The attempt to stop and backout the rolling upgrade may or may not succeed, depending at what point in the rolling upgrade that the request to stop occurs. If a backout of the rolling upgrade is possible, PolicyCenter opens a series of confirmation screens to assist you in correcting issues related to backing out the new configuration.
Start Full Upgrade	Click to open a **Start Full Upgrade** screen. Clicking **Yes** on this screen sets a flag in the database that you intend to start a full database upgrade. You must set this database flag before deploying your new configuration to members of the cluster. If you do not set this flag, PolicyCenter refuses to start and refuses to alter the database. After you complete the full upgrade, PolicyCenter deletes this flag from the database. You must set the flag again before starting a new full upgrade. **Note:** You can also initiate a full PolicyCenter upgrade using the administrative `system_toools` command option `-startfullupgrade`. See "system_tools command" on page 429 for details.
Cancel Full Upgrade	Click to attempt to cancel an in-progress full upgrade. If PolicyCenter determines that it is safe to cancel the upgrade, it does so.

See also

- "Review profiler upgrade information" on page 413

The upgrade report

The Server Tools **Upgrade and Version** screen contains the means to view details of a specific upgrade or download the data for local viewing. The downloadable upgrade information contains the following set of linked reports:

Upgrade Instance	Lists information on various upgrade statistics. It is also to possible to download Guidewire Profiler data by clicking the **Raw Profiler Data** link at the bottom of the **Upgrade Instance** screen.
Database Parameters	Lists information on various database parameters, including the database connection pool settings, the database configuration settings, and similar information.

See also

- "The Upgrade and Versions screen" on page 397
- "Understanding data model updates" on page 270

View an upgrade report

Procedure

1. Navigate to the Server Tools **Upgrade and Versions** screen.
2. For the upgrade process that interests you, click one of the following:

Download Details arrow	Downloads a Zip file that contains various types of upgrade information.
View Details icon	Open a pop-up window from which you can view the upgrade reports.

3. If you downloaded the report file, unzip the file into its own directory.
4. Locate file `index.html` and double-click it to open it in a browser.
5. Use the links on the screen to navigate through the linked reports.

Understanding Guidewire software versioning

Guidewire application versioning is a way to label a particular snapshot of a Guidewire application. It is a string label that Guidewire applies to its software. You see this version number primarily in Server Tools, in the **Cluster Members** screen and the **Upgrade and Versions** screens.

Guidewire manages software versioning through internal classes that provide the following information:

- Guidewire application name and version
- Guidewire module name and version

PolicyCenter writes version information to the application log at PolicyCenter start.

Application version numbering

Both the Server Tools **Cluster Members** screen and the **Upgrade and Versions** screens show the PolicyCenter application version as the following sequence of numbers:

```
A.B (a,b,c,d,e)
```

These numbers have the following meaning:

Version #	Meaning
A	Guidewire application version plus the application release build version, for example, 9.0.0.905.
B	Custom version label
a	Platform major version
b	Platform minor version
c	Application major version
d	Application minor version
e	Data model version number

The following example illustrates these concepts.

```
9.0.0.905.20161017 (6,22,11,45,175)
```

Notice that:

- 9.0.0.905 is the application version number (9.0.0) plus the application release build version number (905).
- 20161017 is a custom label that you define in file `customer-version.properties`. In this case, the label indicates the date of a PolicyCenter configuration upgrade. If not defined, this field is empty.
- 175 is an upgrade version number that you define in file `extensions.properties`.

WARNING In a production environment, Guidewire requires that you increment the data model version number whenever you make changes to the data model, before you restart the application server. Otherwise, unpredictable results can occur. Use of the `extensions.properties` file in a development environment is optional.

See also

- “File customer-version.properties” on page 401
- “Create a custom version label file” on page 401
- “Deploy a custom version label file on Tomcat” on page 402
- *Configuration Guide*

File customer-version.properties

Use file `customer-version.properties` to define a custom build version number. If you create this file and populate it correctly, PolicyCenter appends your custom version number to the Guidewire software version label. Most commonly, you use a custom build number to label and track a configuration deployment that you undertake as a rolling upgrade.

File `customer-version.properties` contains the following single property:

```
customer.build
```

The following example illustrates how to use this property:

```
customer.build=20160914-PCF-upgrade
```

PolicyCenter does not contain file `customer-version.properties` in the base configuration. Instead, you must create this file and place it in a location that PolicyCenter recognizes.

See also

- “Understanding Guidewire software versioning” on page 400
- “Create a custom version label file” on page 401
- “Deploy a custom version label file on Tomcat” on page 402

Create a custom version label file

About this task

Guidewire recommends that you place any `customer-version.properties` file that you create in the `res` directory.

Procedure

1. Open Guidewire Studio™.
2. In the **Project** window, expand **configuration→res**.
3. Select **res** and right-click to open the context menu.
4. Select **New→File**.
5. Name the file `customer-version.properties`.
6. Enter a value for `customer.build`, for example:

```
customer.build=20160914-PCF-upgrade
```

7. Save your work.

See also

- "Understanding Guidewire software versioning" on page 400
- "File customer-version.properties" on page 401
- "Deploy a custom version label file on Tomcat" on page 402

Deploy a custom version label file on Tomcat

Before you begin

Before starting this procedure, ensure that you complete all steps in "Create a custom version label file" on page 401.

Procedure

1. Create a Tomcat WAR file:
   a. Open a command prompt in the PolicyCenter installation directory.
   b. Execute the following command:

      ```
      gwb warTomcatDBcp
      ```

      PolicyCenter adds file `customer-version.properties` to the following JAR file in the generated WAR file:

      ```
      WEB-INF/lib/configuration.jar
      ```
2. Deploy the WAR file to the application server.

Result

After starting the server, the Server Tools **Upgrade and Versions** screen shows the custom version label, appended to the standard application version number.

See also

- "Understanding Guidewire software versioning" on page 400
- "File customer-version.properties" on page 401
- "Create a custom version label file" on page 401

Performing server upgrades

From the **Upgrade and Versions** screen, you can take the following actions:
- Initiate a full database upgrade.
- Cancel a full database upgrade.
- Initiate a rolling upgrade.
- Complete a rolling upgrade.
- Back out a rolling upgrade.

Full application database upgrade

For a full database upgrade, Guidewire requires that you first set a database flag to indicate that a full upgrade is in progress. This action signals your intention to perform a full upgrade. After you complete the upgrade of all servers in the cluster, PolicyCenter deletes the database flag automatically. You must set the upgrade flag again before starting a new full upgrade.
It is possible to set the full upgrade in progress flag in the following ways.

PolicyCenter	To set the upgrade flag in PolicyCenter, click **Start Full Upgrade** in the Server Tools **Upgrade and Versions** screen.
System tools	To set the upgrade flag through system tools, use the following command option: `system_tools -startfullupgrade` At least one cluster member must be running in order for you to use this option.
Web services	To set the upgrade flag using web services, call the `SystemToolsAPI` web service method `startFullUpgrade`. At least one cluster member must be running in order for you to use this option.
Java system property	To set the upgrade flag through a Java system property, use the following system property to set the expected date of the upgrade while starting one of the affected servers: `gwb runServer -Dgw.pc.full.upgrade.intended.date=`*`date`* The *`date`* property is the current date in `yyyyMMdd` format.
File `database-config.xml`	To set the upgrade flag on server start, set the `autoupgrade` attribute on the `<database>` element in file `database-config.xml`.

If you encounter a situation in which all cluster members refuse to start because the upgrade flag was not set, you cannot set the upgrade flag through the server. Instead, you must use one of the following methods to start the full upgrade:

- Set the upgrade flag using the Java system property.
- Set the `autoupgrade` attribute in file `database-config.xml`.

Both of these methods provide the necessary permission to perform a full database upgrade of the cluster members.

Rolling upgrade

For a rolling upgrade, Guidewire first requires that you click **Start Rolling Upgrade** in the **Upgrade and Versions** screen (on any cluster member). This action signals your intention to perform a rolling upgrade and sets a rolling upgrade in progress database flag. If you do not set the upgrade flag, PolicyCenter refuses to start a rolling upgrade.

After you complete the upgrade of all servers in the cluster, you must click **Rolling Upgrade Complete** on the **Upgrade and Versions** screen, which removes the upgrade flag. After you do so, it is not possible start a cluster member running the source (old) configuration.

Guidewire permits a rolling upgrade of the individual members of a PolicyCenter cluster under certain conditions only. In effect, the source (old) configuration and target (new) configuration must be compatible in very specific ways.

Thus, during a rolling upgrade, if you mistakenly deploy an incompatible WAR/EAR file to a PolicyCenter server, you can encounter a situation in which the server does not start. This is true whether you have set the rolling upgrade in progress flag. In this case, remove the incompatible WAR/EAR file and deploy a compatible WAR/EAR file before attempting to restart the server.

Backing out a rolling upgrade

During a rolling upgrade of a PolicyCenter cluster , the Server Tools **Upgrade and Versions** screen shows an **Initiate Backout** button. Clicking **Initiate Backout** opens the beginning screen of a back out wizard. Before you begin the process of backing out a rolling upgrade of a PolicyCenter, first shut down any cluster node that is running the new, target, application configuration.

See “Back out a rolling upgrade” on page 403.

Back out a rolling upgrade

It is possible to back out configuration changes on a given server application instance, depending on the state of the configuration deployment on that instance.

About this task

Guidewire does not recommend that you back out an ongoing deployment on a server instance except in extraordinary circumstances. If you do so, you need to test that the server configuration returns to a safe state.

Procedure

1. Open the Server Tools **Upgrade and Versions** screen.
2. Click **Initiate Backout**.
3. In the **Shut Down New Nodes** screen that opens:
   a. Review the screen and verify that there are no application servers that are running the new, target, configuration.

      If there are any servers running the new configuration, the screen lists the server IDs.
   b. After you shut down all of the servers listed on the screen, if any, click **Refresh**.
   c. After PolicyCenter verifies on-screen that there are no cluster servers running the new configuration, click **Continue**.
4. In the **Verify Typekeys** screen that opens:
   a. Review the on-screen table for information on orphaned type keys.
   b. After you fix any listed issues, refresh the screen to verify your fixes.
5. Click **Start back out process**.
6. If the back out operation does not succeed, navigate to the **Upgrade and Versions** screen:
   a. Click the view icon for the attempted upgrade operation:
   b. Review the **Upgrade Instance** report that opens, paying especial attention to the items listed under **Additional Information**.
   c. If any of the listed items contains a clickable link, open that report and resolve any outstanding issues. For example, open the **File Mismatches** link, if available, and resolve any outstanding issues listed on the report.
7. After you resolve any outstanding issues, click **Initiate Backout** on the **Upgrade and Version** screen again.

   If the back out process succeeds, the **Upgrade and Version** screens indicates that the back out operation was successful.

The About Guidewire PolicyCenter pop-up

During the start-up process for Guidewire Studio, PolicyCenter displays an **About Guidewire PolicyCenter** pop-up box for a few seconds. This pop-up contains application information about your PolicyCenter installation.

Copyright information

The pop-up provides the current software copyright information. For more information about Guidewire trademarks, refer to the following Guidewire site:

```
https://www.guidewire.com/legal-notices
```

Application version information

The pop-up lists the following application version information:

- Application version
- Platform version
- Schema version
- Customer version
- Server instance

The Cache Info screens

The Server Tools **Cache Info** screens provide information in both table and chart form of PolicyCenter server cache information. Guidewire recommends that you use this information to help you monitor how well the cache is performing.

See also

- "Server cache tuning parameters" on page 97
- *Configuration Guide*

The Cache Summary screen

The Server Tools **Cache Summary** screen includes the following information:

Max Cache Space (KB)	Maximum amount of space to allot to the global cache.
Stale Time (mins)	Maximum time allowed for an object to be in the cache without a database refresh.
Page Loaded at	Date and time of the last refresh of the data on this screen.

Available actions on this screen

From the **Cache Info** screen, you can do the following:

Edit	Click **Edit** to enter different values for **Maximum Cache Space** and **Stale Time**. If you change these parameters from the **Cache Summary** view, the values you specify apply only to the PolicyCenter server to which you are connected. If you restart the server, your changes are lost. For your changes to persist, edit the their values in file `config.xml`.
Refresh	Click **Refresh** to update the information shown on this screen. This action also updates the date and time values shown for the **Page Loaded at** field.
Download	Click **Download** to download a CSV-formatted file containing detailed cache information. See also "Understanding the cache data report" on page 406.
Clear Global Cache	Click **Clear Global Cache** to clear the cache of all entities. Note, however that the cache always contains some objects to support an active server.

Cache summary graphs

The **Cache Summary** screen provides the following graphs.

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. This graph is not visible if configuration parameter `GlobalCacheDetailedStats` in `config.xml` is set to `false`.
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 This graph is not visible if configuration parameter `GlobalCacheDetailedStats` in `config.xml` is set to `false`.
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.

See also

- *Configuration Guide*

The Historical Performance screen

Note: For the **Historical Performance** screen to be visible, configuration parameter `GlobalCacheDetailedStats` in file `config.xml` must be set to `true`.

The Server Tools **Historical Performance** screen provides the following graphs:

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. In this case, the x-axis represents the average values for each time period during each of the past eight days. This is to allow for comparison of cache behaviors against hourly trends.
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 screen

The Server Tools **Cache Details** screen includes graphs of the following:

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.

Understanding the cache data report

Clicking **Download** on the Server Tools **Cache Info** screen downloads a CSV-formatted file containing information on the current state of the server cache.

In looking at the cache data provided in the downloaded report, Guidewire recommends that you first calculate the cache miss ratio around the time of the performance degradation. The cache miss ratio is the ratio of (misses) / (misses + hits).

The cache miss ratio is a useful metric in that it normalizes the cache values. For example, suppose that you have the following hit and miss cache values that you use to calculate each individual cache miss ratio.

Timestamp	# cache misses	# cache hits	Cache miss ratio
Time 1	12345	2345	12345 / (12345 + 2345) = 0.8404
Time 2	1234	23456	1234 / (1234 + 23456) = 0.4998

The calculation of the cache miss ratio enables you to compare the data in way that is not possible by simply examining the raw data.

High miss ratio

If the miss ratio is high, it is very likely that the performance issue involves the cache. Thus, you need to look at the reasons for the cache misses, especially the following:

- Number of misses because item was evicted when cache was full
- Number of misses because item was evicted due to reaping

If the number of misses is high due to the cache being full, you can try increasing the size of the cache and observing the result. Keep in mind, however, that the cache uses the heap. Thus, increasing the size of the cache can potentially reduce the amount of heap memory available for other computations.

If the number of cache misses is high due to reaping, you can try increasing the reaping time and observing the result. Another approach would be to examine ways to reduce the time since the last access of the cache objects. For example, how long does it take to reuse objects that have already been fetched from the database? Is it possible to hold onto the objects without having to fetch the objects again from the database?

Cache misses and hits

The cache miss count and the cache miss ratio are probably the most important data points in the download cache data report. Essentially, the more cache hits the better, as it means the cache was able to service that many requests.

Each cache miss is expensive. It means that the looked-for object was not in the cache. Regardless of the hit count, a high miss count can indicate a problem with the cache. Keep in mind, however, that the miss count is relative. A cache miss can occur simply because it is the first time a request is made to the cache for an object. Thus, the object is not in the cache. A cache miss can also occur as the sought-after object may have already been evicted from the cache. The downloaded cache report provides data on the various reasons for object eviction from the cache.

Using the assumption that there is a constant cost for each cache miss, it is possible to compare the miss counts directly. The cost of a cache hit is essentially zero, or very, very small compared with the cost of a cache miss. A cache miss may take a 10 millisecond trip to the database to access the data whereas a cache hit is essentially zero time.

The Guidewire Profiler screens

The Server Tools **Guidewire Profiler** screens provide access to a set of tools that are useful in gathering and analyzing information on the runtime behavior and performance of Guidewire PolicyCenter. The Profiler records the time spent in specific processing areas done by the application code, as well as the configured rules and PCF screens. Use of this information can help narrow down issues to the potentially problematic components such as PCF screens, rules, code sections, or workflows.

Profiler code is useful to record the following types of operations and information:

- SQL statements that PolicyCenter is executing
- Parameters passed to those SQL statements
- Row counts
- Name of the currently executing rule

Guidewire recommends that you exercise care in using this feature. Storing too much information can cause the **Profiler** screen to become too cluttered, require more space for storage and, for long-running processes, hold on to too much memory at runtime.

> **Note:** Guidewire Profiler does not collect memory usage statistics. You can use a third-party tool to gather memory usage and garbage collection information.

See also

- "Server memory management" on page 99

Guidewire profiler concepts

There are several concepts that are important to an understanding of how to configure application profiling. The following list describes these concepts.

Concept	Description	More information
Entry point	Refers to the type of interactions that you select to profile.	“Profiler entry points” on page 408
Profiler tag	Alias for a piece of code to profile in the Guidewire application.	“Profiler tags” on page 413
Profiler frame	Contains information on a specific invocation of profiled code.	“Profiler frames” on page 413
Profiler stack	Stores profiling information for a specific thread.	“Profiler stacks” on page 414

See also

- “The Guidewire Profiler screens” on page 407
- “The Profiler Analysis screen” on page 411

Profiler entry points

The Server Tools Guidewire Profiler collects profile data based on the type of interaction with the application that you request to be profiled. Guidewire refers to the different types of possible interactions as entry points, with the entry point type indicating the type of request or action that initiated application processing. The following list describes the various entry point types that you can access from the Guidewire Profiler **Configuration** screen.

Entry point	More information
Web	“Web session profiling” on page 409
Batch process	“Batch processes” on page 117
Work queue	“Work queues” on page 114
Message destination	*Integration Guide*
REST operations	*Integration Guide*
Web service	*Integration Guide*
Startable service	*Integration Guide*
Gosu servlets	*Integration Guide*

Except for Web entry points, Guidewire Profiler stores configuration information in the database. This information is visible to all PolicyCenter servers in the cluster. For a change in configuration to a batch process, work queue, or message destination to take effect, you need to restart that batch process, work queue, or message destination. Any change in configuration can take some time to propagate through the cluster. Also, it may take up to the cache stale time for a configuration change to become visible.

The next time profiling starts for a given entry point, Guidewire Profiler checks whether profiling is enabled for that entry point. If profiling is enabled, Guidewire Profiler records the profiling data in the form of a profiler stack. The Profiler records multiple stacks if the initial thread spawns more threads and the developer profiles the spawned threads. Except for Web profiling, PolicyCenter persists this data database, making it possible to retrieve the data later.

See also

- “Web session profiling” on page 409
- “PolicyCenter Application profiling” on page 409
- “The Profiler Analysis screen” on page 411
- “Ways to view a Guidewire profiler analysis reports” on page 412

Web session profiling

After you enable web profiling in Guidewire Profiler, PolicyCenter records all subsequent round-trips to the server as a separate profiler stack. Unlike other types of entry points, PolicyCenter does not persist the stacks from Web requests to the database. Instead, PolicyCenter stores the stacks from web requests in the user session.

In using web profiling, Guidewire recommends the following:

- Enable the web profiler only as absolutely needed as it degrades performance.
- Start the profiler from the PolicyCenter application screen that you want to profile.
- Log out of the application after you have analyzed or downloaded the profiler data to free the memory used by the profiler.

It is only possible to enable profiling for web entry points for the current session.

Enable web profiling

Procedure

1. Navigate to the Server Tools **Guidewire Profiler** screen.
2. Select the profiling options that you want to use for this profiling session.
3. Restart the associated batch process, work queue, or message destination, if appropriate.
4. Click **Enable Web Profiling for this Session**.

   This action returns you to the application screen from which you started.
5. Exercise the application screens that you want to profile.
6. Press `ALT+SHIFT+P` to return to Guidewire Profiler.
7. Click **Disable Web Profiling** to end the current Web profiling session.

   This action generates the **Web Profiler** screen.

See also

- To understand the various web profile tracing options, see “Profiler trace options” on page 410.
- To understand the **Web Profiler** screen, see “The Profiler Analysis screen” on page 411.
- To download and save this snapshot of application profiling data, see “View uploaded profiler reports” on page 412.

PolicyCenter Application profiling

You enable application profiling in the Server Tools **Guidewire Profiler→Configuration** screen. On the **Configuration** screen, directly below the **Web Profiler** area, is an area labeled **Entry Point Configuration**. This area contains a set of buttons that you click to enable the other types of entry points profiling. These buttons are:

- **Enable Profiling For Batch Process**
- **Enable Profiling For Work Queue**
- **Enable Profiling For Message Destination**
- **Enable Profiling for Rest Operations**
- **Enable Profiling For Web Service**
- **Enable Profiling For Startable Service**
- **Enable Profiling for Gosu Servlet**

Clicking the button for an entry point opens a secondary screen in which you select the specific item to profile.

For example, to enable application profiling for message destinations:

- Click **Enable Profiling for Message Destinations**. This action opens a screen in which you can select the specific message destination to profile.
- Select a specific message destination and click **OK** to return to the **Configuration** screen.

PolicyCenter then adds a message destination row to the **Entry Point Configuration** table, with a different column for each profile trace option. PolicyCenter places an icon in each table cell for that option.

- If a trace option is not available for the chosen entry point, PolicyCenter marks the table cell for that option with a gray circle. You cannot remove or change the gray circle.
- If a trace option is available for the chosen entry point, PolicyCenter marks the table cell for that option with a red X. The red X means that option is available but disabled. Click the red X to enable the option. PolicyCenter replaces the red X with a green check mark, indicating that option is now available.

For each entry point row, PolicyCenter also adds an **Edit Configuration** button at the right-hand end of the row. This button provides an alternative way to enable tracing options, as well as a way to disable tracing for this entry point.

If you disable an application profile entry type, PolicyCenter removes the row for that entry point from the **Entry Point Configuration** table.

See also

- "Profiler entry points" on page 408
- "Profiler trace options" on page 410
- "Web session profiling" on page 409
- "The Profiler Analysis screen" on page 411

Profiler trace options

It is possible to enable different types of trace options in the Server Tools **Guidewire Profiler→Configuration** screen for the various types of entry points. These options are quite expensive to compute. Guidewire recommends that you narrow down your performance issues first before trying these options.

The exact number and type of tracking options available for profiling depend on the following:

- The application server's operating system
- The application server's database type
- The chosen entry point

After you select a specific entry point to profile, PolicyCenter inserts a row for that profiler under **Entry Point Configuration**. For each table row, you can select one or more profiling options. Only those options with a red X in the table cell are available for that entry point. To enable an option, click the red X. PolicyCenter changes the icon to a green check mark indicating that the option is now active. The following table lists the potentially available profiler options.

Trace option	Description
Individual Stacks	Available for work queue entry points only. If checked, this tracing option stores one stack for each 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 if there are only a few items in the queue.
Stack Trace Tracking	Captures the Java stack trace and the PCF trace at the point at which a query executes. Use caution with this tracing option as it can be very performance intensive. As a general rule, Guidewire recommends that you do not enable this tracing option unless there are very specific reasons to use it.
Query Optimizer Tracing	(Oracle) This trace indicates how the database arrived at a specific execution path. This creates a trace file on the database server.
High Resolution Clock	(Microsoft Windows) This tracing option uses the Microsoft Windows 100 nanosecond clock.
Extended Query Tracing	This trace tracks the parse-execute-fetch actions of a SQL statement. If executing against an Oracle database, this tracing option also tracks any wait times associated with the SQL statement, including what the cause of the wait. This stack trace creates a trace file on the database server.

Trace option	Description
Diff DBMS Instrumentation Counters	(Oracle) 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) The Profiler generates a DBMS report if an action exceeds the threshold value set by this option. This tracing option is available only if you first enable **Diff DBMS Instrumentation Counters** tracing. Click **Edit Configuration** and select **Edit DBMS Instrumentation Capture Threshold (millis)** to edit the default value of 0.

See also

- "Profiler entry points" on page 408
- "Web session profiling" on page 409
- "PolicyCenter Application profiling" on page 409
- "The Profiler Analysis screen" on page 411

The Profiler Analysis screen

After you disable the current application profiling session, PolicyCenter generates a **Profiler Analysis** screen, or, in the case of Web profiling, a **Web Profiler** screen.

The **Profiler Analysis** screen of the Guidewire Profiler contains the following regions:

- **Profiler Source** – Lists each profiler run in the current user session. To see the results for a particular run, select it from the **Name** list.
- **Profiler Results** – Provides a means to view different views of the profiler data for the selected run. Selecting a different view type updates the screen data to reflect your choice.

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. See the *Rules Guide*.
Group Frames	Lists frames in each stack aggregated by tag and presented in order of total aggregate time on the stack.
Group Stacks	Presents data similar to that presented in the **Group Frames** view, except Guidewire Profiler aggregates the frames across all stacks in the session instead of by stack.
Stacks Grouped	Lists the stacks in the profiling session grouped by name, along with timing information on each stack group.
Rule Execution	Lists rules that fired during the session. If no rules fired during the session, the **Profiler Result** region contains the message "No profiler stacks found".

See also

- "The Guidewire Profiler screens" on page 407
- "View uploaded profiler reports" on page 412
- *Rules Guide*

Save a profiler analysis report

About this task

This procedure creates a `.gwprof` file that you can download and store outside of Guidewire Profiler. To view the contents of this file, you must upload the `.gwprof` file back into Guidewire Profiler.

Procedure

1. Navigate to the Server Tools **Guidewire Profiler→Profiler Analysis→Profiler Analysis** screen.
2. Select the profiler entry point, for example, **Batch Processes**.
3. In the **Profiler Source** area at the top of the screen, select the report that interests you.
4. In the **Profiler Result** area, select a value from the **View Type** drop-down list.
5. Click **Download**.
6. In the dialog that opens, select **Save File** and click **OK**.

See also

- "View uploaded profiler reports" on page 412

Ways to view a Guidewire profiler analysis reports

It is possible to view Guidewire Profiler analysis reports in any of the following ways.

View type	For more information
View current report in **Profiler Analysis** screen	"The Profiler Analysis screen" on page 411
Search for historical report and view in **Profiler Analysis** screen	"View historical profiler reports" on page 412
Upload historical report and view in **Profiler Analysis** screen	"View uploaded profiler reports" on page 412

See also

- "The Guidewire Profiler screens" on page 407

View historical profiler reports

Procedure

1. Navigate to the Server Tools **Guidewire Profiler→Profiler Analysis→By Time Range** screen.
2. Click **Search**.
   The screen shows two calendar date pickers.
3. Enter a start and stop date in which to search for existing profiler analysis reports.
   PolicyCenter prints the results of the search to the screen:
   - If PolicyCenter finds no existing profiler reports that occurred within the specified time frame, it prints a message to that effect.
   - If PolicyCenter does find one or more profiler reports that occurred within the specified time frame, it lists the reports.
4. In the **Profiler Source** area at the top of the screen, select the report that interests you.
5. In the **Profiler Result** area, select a value from the **View Type** drop-down list.

View uploaded profiler reports

Before you begin

The file to upload must be a file with a `.gwprof` extension downloaded previously from Guidewire PolicyCenter.

Procedure

1. Navigate to the Server Tools **Guidewire Profiler→Profiler Analysis→Saved File** screen.
2. In the **Restore Snapshot** field, browse to find a `.gwprof` file for upload.
3. Click **OK**.

Result

The saved profiler data loads in the **Saved File** screen. which then becomes the **Profiler Analysis** screen. If you navigate away from this screen, Guidewire Profiler deletes the uploaded data from the screen.

See also

- "Save a profiler analysis report" on page 412

Review profiler upgrade information

About this task

The Guidewire Profiler **Upgrade** screen provides profiler analysis of upgrade operation on the PolicyCenter server. The **Upgrade** screen provides information on the database upgrade at initial server start and all subsequent upgrade operations.

Procedure

1. Navigate to the Server Tools **Guidewire Profiler→Profiler Analysis→Upgrade** screen.
2. In the **Profiler Source** area at the top of the screen, select the report that interests you.
3. In the **Profiler Result** area, select a value from the **View Type** drop-down list.

See also

- "The Upgrade and Versions screen" on page 397
- "The Guidewire Profiler screens" on page 407
- "The Profiler Analysis screen" on page 411

Profiler tags

Profiler tags represents sections of code that Guidewire Profiler can profile. A profiler tag is an alias for a piece of code in the Guidewire application for which you want to gather performance information.

The code represents profiler tags by instances of the `gw.api.profiler.ProfilerTag` class. The constructor for the `ProfilerTag` takes a `String` parameter defining the `ProfilerTag` name.

Always create a static final `ProfilerTag` object and preserve it. If you attempt to create more than one instance of the same `ProfilerTag` object, PolicyCenter generates a warning message in the application log that is similar to the following:

```
WARN Duplicate tag name: tag_name
```

Profiler frames

A profiler frame contains information corresponding to a specific invocation of profiled code, such as its start and finish times.

In each Profiler session:

- Whenever the code calls `push` on the profiler stack, Guidewire Profiler creates a profiler frame and pushes the frame onto the stack.
- Whenever the code calls `pop` on the profiler stack, Guidewire Profiler removes the profiler frame from the stack. The Profiler continues to stores the frame information, however, so as to make the information available for future examination.

The code represents Profiler frames by instances of `gw.api.profiler.ProfilerFrame`.

See also

- "Understanding properties and counters on a frame" on page 415

Profiler stacks

A profiler stack stores profiling information for a specific thread. A profiler stack implements the standard `push` and `pop` functionality of a stack. The `push` and `pop` actions 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 PolicyCenter is currently executing. The code represents Profiler stacks 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.

Using custom profile tags with Guidewire Profiler

It is possible to profile custom code in the Server Tools **Guidewire Profiler** by creating your own custom profile tags. To define a custom Profiler tag, create a globally-accessible Gosu class using the Java-based `PCProfilerTag` class as a model. Guidewire recommends that you add all custom profiler tags to this Gosu class.

The following sample code illustrates how to create a custom Gosu class for custom profiler tags.

```
package gw.profiler

 uses gw.api.profiler.ProfilerTag

 class MyProfilerTags {
  public static final var MY_TEST_TAG1 = new ProfilerTag("MyTestProfiler1")
  public static final var MY_TEST_TAG2 = new ProfilerTag("MyTestProfiler2")
  public static final var MY_TEST_TAG3 = new ProfilerTag("MyTestProfiler3")
  //..

   private construct() {
    // Do not instantiate}
  }
}
```

To profile a block of custom code, use the following pattern to push and pop profiling information onto the profiler stack.

```
uses gw.api.profile.Profiler
...

 ProfilerFrame frame = Profiler.push(ProfilerTag.MY_TEST_TAG1)
try {
  // CODE TO PROFILE
} finally {
  Profiler.pop(frame)
}
```

See also

- "Understanding properties and counters on a frame" on page 415

Profiling spawned threads

Some processes spread their workload across multiple threads. If you want to use Guidewire Profiler to profile these threads, use the following pattern:

```
gw.api.profiler.Profiler.createPotentiallyProfiledRunnable(ProfilerTag entryPointTag,
      String entryPointDetail, GWRunnable block)
```

This generates a new `Runnable` object that executes the given block. This `Runnable` object profiles the block if the calling thread is also being profiled. If this is the case:

- The Profiler associates the stack for that thread with the stack of the calling thread.
- The Profiler persists that thread along with the stack of the calling thread.

See the Javadoc for the `Profiler.createPotentiallyProfiledRunnable` method for more details.

Understanding properties and counters on a frame

Guidewire profiler frames can hold custom properties and counters that provide more information about system events. Consider the following example:

```
uses gw.api.profiler.Profiler
uses gw.api.profiler.ProfilerTag

 public class MyProfilerTags {

  public static var paramValue : String
  public static var ctrValue : Integer

   var frame = Profiler.push(MyProfilerTags.myProfilerTag)

  public function profileCode() {

     public static var myProfilerTag: ProfilerTag = new ProfilerTag("MY_TAG")

    try {

      //Some paramter value, set by method code...
      var param = "some parameter value"

      //Some counter value, set by method code
      var ctr = 5

      frame.setPropertyValue("PARAMETER", param)
      frame.setCounterValue("COUNTER", ctr)

      } finally {

      Profiler.pop(frame)

    }

  }

 }
```

After the sample code pops a profiler frame off the stack, the frame contains information about the calculated values of `PARAMETER` and `COUNTER`. The Server Tools **Profiler Analysis** screen then shows these values as well.

See also

- “The Guidewire Profiler screens” on page 407
- “The Profiler Analysis screen” on page 411
- “Using custom profile tags with Guidewire Profiler” on page 414

Understanding the Guidewire Profiler API

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 also

- *Integration Guide*

Profiler-related configuration parameters

Guidewire provides several profiler-related configuration parameters in `config.xml`. The first two configuration parameters in the list refer to SQL Server databases only.

Parameter	For more information
`IdentifyQueryBuilderViaComments`	*Configuration Guide*
`IdentifyORMLayerViaComments`	*Configuration Guide*
`ProfilerDataPurgeDaysOld`	*Configuration Guide*

See also

- "The Guidewire Profiler screens" on page 407
- *Gosu Reference Guide*

The Product Model Info screen

The Server Tools **Product Model Info** screen contains a single **Reload Availability** button. If you click this button, PolicyCenter attempts to reload availability data from the directory specified in the `ExternalProductModelDirectory` parameter defined in `config.xml`.

> **Note:** Guidewire disable the **Reload Availability** button during a rolling upgrade. The button does not become active until a user clicks **Rolling Upgrade Complete** in the Server Tools **Upgrade and Versions** screen.

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.

To access the **Product Model Info** screen:

- The server must be in development mode.
- The `EnableInternalDebugTools` parameter in `config.xml` must be set to `true`.
- The logon user role must have the **View ProductModelInfo tools screen** permission. The code for this permission is `toolsProductModelInfoview`.

See also

- *Product Model Guide*
- *Configuration Guide*

chapter 26

Internal tools

Guidewire provides internal tools to assist you with certain administrative tasks.

WARNING Guidewire does not support the use the tools found in the **Internal Tools** screens. Guidewire provides these tools for use during development only. Guidewire does not support the use of these tools in a production environment. Use these tools at your own risk.

See also

- "Server tools" on page 353

Accessing the PolicyCenter internal tools

The server must be in development mode and you must have the `internaltools` permission to access the **Internal Tools** screens. To access the **Internal Tools** screens, press `ALT+SHIFT+T` on any PolicyCenter screen after you log into the application as a user with the necessary role permission.

See also

- "Server modes" on page 83

The Reload screen

The **Reload** screen is useful while you develop a configuration. From this screen 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_`*`LanguageCode`*`.properties` file for the locale.

The Server Performance screen

The **Server Performance** is useful if in tracking and viewing PolicyCenter application server activity.

Toolbar buttons

The **Server Performance** screen contains the following row of buttons at the top of the screen. The button labels are generally self-explanatory.

Button	Action
Enable or **Disable**	Click to enable or disable the tracking of server performance. The button label changes depending on whether server performance tracking is enabled or disabled.
Refresh	Click to update the information shown in the summary table.
Clear Results	Click to remove all information from the summary table.
Export Summary to CSV	Click to export the information shown in the summary table in CSV (comma-separated values) format. The default file name for the download file is `ServerPerformanceSymmaryExport.csv`. The download file contains more information than is shown in the summary table.
Export Raw Data to CSV	Export the underlying data that creates the view shown in the summary table in CSV (comma-separated values) format. The default file name for the download file is `ServerPerformanceRawDataExport.csv`. The download file contains more information than contained in the summary table file.

The Server Performance summary table

The summary table column labels are generally self-explanatory:

- The **Page** column lists each area of PolicyCenter application that the server accesses in alphabetical order.
- The **Requests** column lists the number of times that the server accesses a particular application area.
- The remaining columns lists various times it takes to complete a task, or a total of the completed times.

The Testing System Clock screen

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. Use this tool to move the system time forward as necessary to determine if a process completes correctly. You cannot set the system time to a time before the current time.

IMPORTANT Use the system clock plugin for testing purposes only. You can only adjust the system clock if the PolicyCenter server is in development or test mode.

You must implement and configure the `TestingClock` plugin to use this tool.

See also

- See the *Integration Guide* for implementation details.

The PC Sample Data screen

The **PC Sample Data** screen is for loading sample data into PolicyCenter for development purposes only. Guidewire does not support this tool for a production environment.

See also

- *Installation Guide*

The Free-text Search screen

The **Free-text Search** screen helps you manage the Guidewire Solr Extension, a full-text search engine, during development. The screen is an alternative to the free-text batch load command. The **Free-text Search** screen 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** screen 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** screen, you must run the PolicyCenter application in development mode. The application hides the screen whenever you run the application in production mode.

The **Free-text Search** screen 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** screen 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

- “Run the free-text batch load command” on page 318
- *Installation Guide*
- *Configuration Guide*

The Archiving Test screen

PolicyCenter provides an Internal Tools **Archiving Test** screen that you can use to test archiving on a job or policy term. This screen is only available if the value of configuration parameter `ArchiveEnabled` is `true`.

Use this tool during development to see the effect of Archive Policy Terms batch processing on a selected job or policy term. From this screen, you can do the following:

Action	Description
Archive term by policy transaction number	Enter the job ID of the policy term that you want to archive. For validation: • If you chose to skip validation, PolicyCenter archives the policy term based on the current configuration, including the `IPCArchviginPlugin` implementation. • If you do not skip validation, PolicyCenter archives the policy term regardless of whether there are open jobs or whether the server time has reached the date set by `PolicyTerm.NextArchiveCheckDate`. Clicking **Archive** does not run Archive Policy Term batch processing. Instead, the tool reads the current configuration parameter values and runs the `IPCArchivingPlugin` plugin implementation.
Archive term by policy number and term	Enter the policy and term number of the policy term that you want to archive. This action behaves in a similar fashion to that of **Archive term by policy transaction number**.
Flush other work queues	PolicyCenter cannot archive a policy term if that policy term is associated with one or more workflows. To more effectively archive policy terms, Guidewire recommends that you run this command to clean up workflows before archiving.

Action	Description
	This command runs the following batch processing types: • Purge Workflow • Purge Workflow Logs • Workflow
Run archiving batch process	Click **Run** to start the Archive Policy Term batch processing. This action is the same as running Archive Policy Term batch processing from the **Server Tools→Work Queue Info** screen.

See also

- "Archive Policy Term work queue" on page 133
- "Purge Workflow batch process" on page 153
- "Purge Workflow Logs batch process" on page 153
- "Workflow work queue" on page 157

The Personal Data Destruction Tests screen

PolicyCenter includes a tool to help you test data destruction on a contact, policy, or account and test your implementation of data destruction. You can use this tool during development.

IMPORTANT Do not include this tool in a production system.

To access this tool, log in to PolicyCenter, press `Alt+Shift+T`, and select **Internal Tools→Personal Data Destruction Test**.

The tool enables you to test either data purging or data obfuscation.

- Purge

  Enter a contact public ID, an account number, or a policy number or policy public ID and then:
  - Click **Preview Purge** to see the domain graph nodes that the code traverses and the disposition for each node.

    The disposition is the return value for the `shouldDestroy`*`Object`* methods you defined in your plugin implementation class for the `PCPersonalDataDestruction` plugin.
  - Depending on the object you want to test, click **Purge Contact**, **Purge Account**, or **Purge Policy** to attempt to purge all objects in the graph.
- Obfuscation

  You can test obfuscation either for a user, by user name, or for a user contact, by public ID. You can load the user or user contact first to see if your entry is valid before you test obfuscating it.

chapter 27

Command prompt tools

PolicyCenter includes a number of administrative tools as command prompt tools that you can use for help with administrative tasks on your PolicyCenter server.

Note: For tools that build PolicyCenter, see the *Installation Guide*.

About the administration tools

PolicyCenter provides a set of tools that you can use to perform administrative tasks directly from a command prompt. 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. These tools all execute against a running PolicyCenter instance.

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 actively running.

User credentials

The PolicyCenter administration command prompt tools all require that you enter the password of an administrative user for the tool to work. The use of a user name is optional.

User name

PolicyCenter does not require the use of a user name for an administration command. The use of a user name is optional:

- If you do supply a value for the `-user` parameter, it must be the user name of a user with administrative privileges.
- If you do not supply a value for the `-user` parameter, the command defaults to user `su` (in the base configuration) and you must supply the password for that user.

Password

PolicyCenter expressly requires the use of a password of a user with administrative privileges for an administration command. If you do not supply a value for the `-password` parameter, the execution of the command fails

Masking user credentials

In the administration command, it is possible mask the input of the user name and password. To do so, insert a dash in the command in place of the actual user name or password, for example:

```
system_tools -ping -user - -password -
```

Before PolicyCenter executes the tool script, it prompts you to enter the missing information. If you chose to mask the user name and password information:

- The PolicyCenter application does not echo the entered user name or password back to the screen.
- The operating system, either Windows or Linux, does not store the plain text information in the command prompt buffer and command history.

Accessing administration tool help

To access help for any of the PolicyCenter command prompt tools, enter `-help` after the tool name. For example, enter the following at the command prompt to generate a list of command options with a description of each option for the `import_tools` administrative tool:

```
import_tools -help
```

Administrative tool command syntax

The administrative tools command descriptions use the following command syntax.

`-option`	All command options start with a minus sign (`-`). Command options are either mandatory or optional. See the following discussion.
`|`	An upright bar indicates a Boolean OR. For example, `A | B | C` means `A` or `B` or `C`.
`{ ... }`	A set of curly braces indicates a set of mutually exclusive choices. You must one chose (and only one) item from a set of choices. For example, `{ A | B | C }` indicates you must choose either `A` or `B` or `C`, but not more than of one the listed options.
`arguments`	Specifies the arguments required by a tool option such as a file name or directory, for example, **import_tools** `... -import` *`file`*.
`...`	A series of dots after the argument indicates that you can enter multiple items of the same type. For example, `-import` *`file`* `...` indicates that you can enter multiple file names (*`file`*) after the `-import` argument.
`[ ... ]`	A set of square brackets indicates that the argument is optional. For example, `[-user]` indicates that the command permits you to set a user value (`-user`), but does not require that you set this value. In contrast, an argument not enclosed in square brackets indicates that an argument is mandatory. For example, for all the administrative commands, the `-password` argument is mandatory. Thus, the command syntax does not surround the `-password` argument by square brackets as the argument is mandatory.

PolicyCenter command prompt tools summary

The following list provides a summary of the PolicyCenter administration tools available from a command prompt.

Tool	Description
"data_change command" on page 423	Provides a mechanism for making changes to code on a running production server. **WARNING** Only use the `data_change` command under extraordinary conditions, with great caution, and upon advice of Guidewire 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 the data change on a production server.
"import_tools command" on page 424	Provides a set of utilities for loading XML-formatted data into PolicyCenter.
"maintenance_tools command" on page 426	Provides a set of utilities for performing maintenance operations on the server (for example, running escalation/exception rules, calculating statistics, and more.)

Tool	Description
"messaging_tools command" on page 427	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 command" on page 429	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.)
"table_import command" on page 435	Provides the means for importing tables into the database.
"template_tools command" on page 438	Provides help in converting between template versions.
"workflow_tools command" on page 439	Provides the means to manage user workflows in the system.
"zone_import command" on page 439	Provides the means to load zone data from a file to a staging table.

data_change command

PolicyCenter provides a tightly constrained system for updating data on a running production server. Guidewire calls this mechanism the Production Data Fix tool. The Production Data Fix tool uses either the `data_change` command option, or, the `DataChangeAPI` web service to make changes to the production data.

Because the `data_change` command allows arbitrary execution of data, Guidewire strongly recommends that you carefully control the ability to create and run code on a production server. The user who runs this command must have permission `wsdatachangeedit`.

WARNING Only use the Production Data Fix tool under extraordinary conditions, with great caution, and upon advice of Guidewire 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 the data change on a production server.

```
data_change -help
data_change -password password [-server url] [-user user] {
            -edit refID -gosu filepath [-description desc] |
            -discard refID |
            -status refID |
            -result refID }
```

See also

- For a description of how and when to use the `data_change` command to change data on a running production server, see "Production data fix tool" on page 321.
- For a description of how to use the `DataChangeAPI` web service, see "Data change web service reference" on page 327.

data_change command options

You can use any of the following options with the `data_change` command. You must always supply the `-password` option.

WARNING Only use the Production Data Fix tool under extraordinary conditions, with great caution, and upon advice of Guidewire 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 the data change on a production server.

Option	Description
`-description` *`desc`*	Human-readable description (*`desc`*) of the change. Include this option with the `edit` option. For testing, the description is optional. For production use, include the description. Put quotes around the description to permit space characters in the description.
`-discard` *`refID`*	Instruction to discard a data change that you already registered. You must supply a data change reference ID (*`refID`*). You cannot discard a data change that was already run.
`-edit` *`refID`*	Instruction to create a new data change or edit an existing data change. You must supply a unique reference ID (*`refID`*) 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, you must: • Include the `-gosu` option to include your Gosu data change code • Include the `-description` argument to provide a description
`-gosu` *`filepath`*	Full path name (*`filepath`*) to a Gosu script. You must include this option with the `edit` argument. You can use a full path name, or a relative path that is relative to the current working directory.
`-password` *`password`*	Password (*`password`*) to use to connect to the server. PolicyCenter requires the password.
`-result` *`refID`*	Result of a data change that you already registered. You must supply a data change reference ID (*`refID`*). If a user attempted to run it and there were parse errors, the results include the errors.
`-server` *`url`*	Specifies the PolicyCenter host server URL. Include the port number and web application name, for example: `http://servername:8180/pc`
`-status` *`refID`*	Status of a data change that you already registered. You must supply a data change reference ID (*`refID`*). This option prints the status of the data change, which is one of the following: • `Open` • `Discarded` • `Executing` • `Failed` • `Completed`
`-user` *`user`*	User (*`user`*) to use to run this process.

import_tools command

```
import_tools -help
import_tools -password password [-server url] [-user user] {
           -import filename1, filename2 ... [-charset charset] [-dataset dataset]
                [-ignore_all_errors] [-ignore_null_violations]
                [ { -output_csv filename | -output_xml filename } ] | -privileges }
```

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. PolicyCenter does not throw concurrent data change exceptions if the imported records overwrite existing records in the database.

Data that you import into PolicyCenter through the use of `import_tools` is immediately available. You do not need to restart the PolicyCenter server for the changes to take effect.

IMPORTANT Guidewire supports using the `import_tools` command to import administrative data only.

IMPORTANT The `MaximumFileUploadSize` parameter in `config.xml` must exceed the size of any file that you attempt to import. The `MaximumFileUploadSize` parameter value is in megabytes (MB). The base configuration default value of `MaximumFileUploadSize` is 20 MB.

See also

- "Ways to import administrative data" on page 297
- "About the import directory" on page 298
- "Using tools to import administrative data" on page 307
- *Integration Guide*

import_tools command options

You can use any of the following options with the `import_tools` command. You must always supply the `-password` option.

Option	Description
`-charset` *`charset`*	Character set encoding (*`charset`*) for the files to import. If this option is null, PolicyCenter sets the default character encoding to UTF-8. See also "Character set encoding for file import" on page 300.
`-dataset` *`integer`*	Integer value (*`integer`*) representing the dataset to import from a CSV-formatted file, for example: `RolePrivilege,0,`**default_data:3**`,abdelete,audit_examiner` PolicyCenter orders datasets by inclusion. The number of the smallest dataset is always 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.
`-ignore_all_errors`	Causes the tool to ignore any errors in a CSV-formatted input file.
`-ignore_null_violations`	Causes the tool to ignore violations of null constraints in a CSV-formatted input file.
`-import` *`filename1, filename2, ...`*	Imports administrative data from one or more CSV (comma-separated values data) files or XML files. It is possible to provide a list of file names in a separate file. To do so, create a file that contains a comma-separated list of files names. Prefix an @ character to the name of the list file, for example: `-import @files.lst` To convert data using the `-output_csv` or `-output_xml` options, provide only a single file name.
`-output_csv` *`filename`*	If used with the `-import` option, outputs comma-separated values to the specified file and then stops processing. PolicyCenter imports no data into the server. Use this option to convert XML input files to CSV-formatted output files.
`-output_xml` *`filename`*	If used with the `-import` option, outputs XML to the specified file and then stops processing. PolicyCenter imports no data into the server. Use this option to convert CSV input files to XML-formatted output files.

Option	Description
`-password` *`password`*	Password to use to connect to the server. PolicyCenter requires the password value.
`-privileges`	Adds the role privileges contained in file `roleprivileges.csv` in the Guidewire Studio™ `modules/configuration/config/import/gen` folder to those roles that already exist in the database. See also "About adding admin data after initial server startup" on page 300.
`-server` *`url`*	Specifies the PolicyCenter host server URL. Include the port number and web application name, for example: `http://servername:8180/pc`
`-user` *`user`*	User to use to run this process.

maintenance_tools command

```
maintenance_tools -help
maintenance_tools -password password [-server url] [-user user] {
                 -processstatus process |
                 -startprocess process [-args arg1, arg2, ...] |
                 -terminateprocess process |
                 -whenstats }
```

The `maintenance_tools` command starts, terminates, or retrieves the status of a PolicyCenter process. For a list of processes that the `maintenance_tools` command can start, see "The work queue scheduler" on page 120.

maintenance_tools command options

You can use any of the following options with the `maintenance_tools` command. You must always supply the `-password` option.

Option	Description
`-args` *`arg1 arg2`* `...`	Arguments to use while starting a process. Use only with `-startprocess`. If you have multiple arguments, separate each one with a space. The command does not validate the provided arguments. To use arguments with custom batch processes, see the *Integration Guide*, especially the following method: `ProcessesPlugin.createBatchProcess(type, args)`
`-password` *`password`*	Password (*password*) to use to connect to the server. PolicyCenter requires the password.
`-processstatus` *`process`*	Returns the status of a batch process. For the *process* value, specify a valid process name or a process ID. For work queues, this option returns the status of the writer process. It does not check whether additional work items remain in the work queue. Thus, it is possible for the process status to report completion after the writer finishes adding items to the work queue while the work queue contains unprocessed work items.
`-server` *`url`*	Specifies the PolicyCenter host server URL. Include the port number and web application name, for example: `http://`*`serverName`*`:8180/pc`
`-startprocess` *`process -args...`*	Starts a new batch process. For the *process* value, specify a valid process code. See also `-args`. For a list of batch process codes, including work queue writer processes, see "Work queues and batch processes, a reference" on page 130.
`-terminateprocess` *`process`*	Terminates a batch process. For the *process* value, specify a valid process name or a process ID.

Option	Description
	It is not possible to terminate single phase processes using this option. Single phase processes run in a single transaction. Thus, there is no convenient place to terminate the process. See “Work queues and batch processes, a reference” on page 130 to determine if it is possible to terminate a process.
`-user` *`user`*	User (*`user`*) to use to run this process.
`-whenstats`	Reports the last time PolicyCenter calculated statistics on the server.

messaging_tools command

```
messaging_tools -help
messaging_tools -password password [-server url] [-user user] {
                -config destinationID |
                -purge date |
                -restart -destination destinationID [-wait wait] [-retries retries]
                      [-initial initial] [-backoff backoff] [-poll poll] [-threads threads]
                      [-chunk chunk] |
                -resume destination destinationID |
                -resync -destination destinationID -account -accountID |
                -retry messageID |
                -retrydest destinationID |
                -skip messageID |
                -statistics destinationID |
                -suspend destinationID }
```

You use the `messaging_tools` command to manage a message destination from the command prompt. To do so, you must know the message destination ID. The person who creates the message destination assigns this ID. You create and configure message environments and destinations in file `messaging-config.xml`. Access `messaging-config.xml` in Guidewire Studio at the following location:

configuration→config→Messaging

messaging_tools command options

You can use any of the following options with the `messaging_tools` command. You must always supply the `-password` option.

Option	Description
`-account` *`accountID`*	Use to specify the account ID (*`accountID`*) of the account to re-synchronize. See `-resync`.
`-config -destination` *`destinationID`*	Returns the configuration for a message destination.
`-destination` *`destinationID`*	Specifies a message destination (*`destinationID`*).
`-password` *`password`*	Password (*`password`*) to use to connect to the server. PolicyCenter requires the password.
`-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 option 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:
	• *`destinationID`* – The destination ID of the destination to restart.
	• *`wait`* – the number of seconds to wait for the shutdown before forcing it.
	• *`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.
	• *`poll`* – 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 the *Integration Guide* for details on how the polling interval works. If your performance issues primarily relate to many messages for each primary object for each 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` option to configure the number of sender threads for safe-ordered messages. PolicyCenter ignores this setting for non-safe-ordered messages, as PolicyCenter uses one thread for each destination for these types of messages. 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 the *Integration Guide*.
	• *`chunk`* – number of messages to read in a chunk.

`-resume -destination` *`destinationID`*	Resumes the operation of the specified message destination.

`-resync -destination` *`destinationID`*
`-account` *`accountID`*
	Re-synchronizes an account with specified ID against a specific message destination. Use `-destination` and `-account` 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 for retry if the error at the destination system is temporary and the message destination does not have an automatic retry mechanism. For instance, the message is a candidate for retry if the destination contains a locked record and refuses the update.
`-retrydest` *`destinationID`*	Retries all retryable messages for a message destination.
`-server` *`url`*	Specifies the PolicyCenter host server URL. Include the port number and web application name, for example: `http://`*`serverName`*`:8180/pc`
`-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 option if the destination system is going to be shut down or to halt sending while PolicyCenter processes a daily batch file.
`-user` *`user`*	User (*`user`*) to use to run this process.

system_tools command

```
system_tools -help
system_tools -password password [-server url] [-user user] {
             -cancelshutdown serverId |
             -cancelupdatestats |
             -checkdbconsistency [-tableselection tblSelection
                   -checkTypeSelection checkTypeSelection] |
             -completefailedfailover type componentId |
             -components |
             -daemons |
             -dbcatstats [regularTables stagingTables typelistTables] |
             -getdbccstate |
             -getdbstatisticsstatements |
             -getincrementaldbstatisticsstatements |
             -getPerfReport reportID |
             -getupdatestatsstate |
             -listPerfReports [numReports] |
             -loggercats |
             -maintenance |
             -mssqlPerfRpt [n1 n2 collectStatistics] |
             -multiuser |
             -nodefailed serverId [-evenifincluster -filepath filepath] |
             -nodes |
             -oraListSnaps numstats |
             -oraPerfReport beginSnapshotID endSnapshotID probeVDollarTables |
             -ping |
             -recalcchecksums |
             -reloadloggingconfig |
             -requestcomponenttransfer type componentId targetOwner |
             -scheduleshutdown serverId [-terminatebatchprocesses | -shutdowndelay minutes] |
             -sessioninfo |
             -sqlListIntervals numIntervals |
             -startfullupgrade |
             -updatelogginglevel loggername logginglevel |
             -updatestatistics description update |
             -verifyconfig filepath |
             -verifydbschema |
             -version }
```

See also

- “Managing database statistics using system tools” on page 287
- “Database Statistics work queue” on page 138
- *Integration Guide*

system_tools command options

You can use any of the following options with the `system_tools` command. You must always supply the `-password` option.

Option	Description
`-cancelshutdown serverId`	Cancels the planned shutdown of the server specified by `serverId`.
`-cancelupdatestats`	Cancels the process that is updating database statistics if running. Use the following option to verify the process state: `-getupdatestatsstate`
`-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 during a time period of low activity on the database server.

Option	Description
	The `-checkdbconsistency` option has two optional arguments: • *`tblSelection`* • *`checkTypeSelection`* Specify the *`tblSelection`* argument as one of the following: • `all` – Run consistency checks on all tables. • *`tableName`* – The name of a single table on which to run checks. • `tg.`*`tableGroupName`* – The name of a table group. Use the `<database>` element in `database-config.xml` to define table groups. For more information, see the *Installation Guide*. • *`@fileName`* – The name of a file that contains 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. Specify the *`checkTypeSelection`* argument as one of the following: • `all` – Run all consistency checks on the specified tables. • *`checkName`* – The typecode of a single consistency check to run. • *`@fileName`* – The name of a file 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. To run consistency checks from PolicyCenter, use the **Server Tools Consistency Checks→Info Page** screen, described in “The Consistency Checks screen” on page 364. For more information, see “Database consistency checks” on page 272.
`-completefailedfailover` *`type componentId`*	Manually completes component failover for a failed component. You must supply the component type (*`type`*) and component ID (*`componentId`*).
`-components`	Provides information about the components that exist on each PolicyCenter server in the cluster. The report contains the following information for each component: • Component type • Component code • Component state • Component start date and time • Server ID of the server instance on which the component exists • Component ID The report information is similar, but not identical, to the cluster information available from the Server Tools **Cluster Members** and **Cluster Components** screens. See “The Cluster Members screen” on page 391 for information on these screens.
`-daemons`	Sets the server to the `daemons` run level. For information about the various run levels, see “Server run levels” on page 84.
`-dbcatstats`	Used with no arguments, the option returns a ZIP file of database catalog statistics info for all the tables in the database. The `-dbcatstats` option takes the following optional arguments: • *`regularTables`* • *`stagingTables`* • *`typelistTables`* This option, used with three arguments, returns a ZIP file of database catalog statistics info for the specified tables. Specify each of the arguments as one of the following: • `all/none` – Select all or none of the tables of this type • `<`*`tableName`*`>` – The name of a single table of this type • `@<`*`fileName`*`>` – The name of a file with one or more valid table names of this type entered in comma-separated values (CSV) format.

Option	Description
	For example, `-dbcatstats none none all` returns database catalog statistics information for all the typelist tables. You must specify either no arguments or all three arguments if you use this command option. You can specify the target destination for the database catalog statistics ZIP file by adding the `–filepath` *`filepath`* option. 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 option, try reducing the number of tables on which to gather statistics by using the arguments listed previously. For information about configuring database statistics generation, see "Understanding database statistics" on page 285.
`-evenifincluster [-filepath` *`filepath`*`]`	Consider the cluster member as failed even if it is still in the cluster. Use only with the `-nodefailed` option. The `-filepath` parameter sets the location (*`filepath`*) for an optional report. **IMPORTANT** This command option overrides an important safety check on the server. Use this command option in certain defined circumstances only. See `-nodefailed` for details.
`-getdbccstate`	Returns the status of any currently executing database consistency checks, Processing or Completed, for example.
`-getdbstatisticsstatements`	Retrieves the list of SQL statements to update database statistics and prints the list to the console. See "Understanding database statistics" on page 285.
`-getincrementaldbstatisticsstatements`	Retrieves the list of SQL statements to update database statistics for tables exceeding the change threshold. Prints the list to the console. The `incrementalupdatethresholdpercent` attribute of the `<databasestatistics>` element in `database-config.xml` defines the change threshold. See "Understanding database statistics" on page 285.
`-getPerfReport` *`reportID`*	Downloads the performance report with the specified *`ID`* to a local directory. You can retrieve a list of available performance report IDs by running the `-listPerfReports` command option. To download the report file to a directory other than the default directory, add the `-filepath` option to the `-getPerfReport` command option.
`-getupdatestatsstate`	Returns the state of the process running the statistics update.
`-listPerfReports` *`numReports`*	Lists IDs and other information for available database performance reports. You can specify an optional integer (*`numReports`*) to specify the number of available downloads to list, ordered starting with the most recent. If unspecified or `0`, this command lists all available downloads. 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 `-getPerfReport` *`ID`* command option.
`-loggercats`	Displays the available logging categories.
`-maintenance`	Sets the server to the `maintenance` run level. For information about the various run levels, see "Server run levels" on page 84.

Option	Description
`-mssqlPerfRpt` *`n1 n2 collectStatistics`*	Generate a SQL Server performance report defined by the option parameters, which have the following meaning: • *`n1`*, *`n2`* - One, or two, interval IDs that define the time interval of the report. A single ID indicates a report for that time period only. Two IDs indicate a time range. If you do not supply at least one interval ID, PolicyCenter does not generate Query Store statistics. • *`collectStatistics`* - A Boolean value that determines whether PolicyCenter includes database statistics (`true`) in the output report.These statistics are the same that you see in the Server Tools **Database Statistics** screen. After PolicyCenter generates the report, the **SQL Server Performance Report** screen shows the time interval used for the report in the report description. If you do not know the interval IDs to use, run the `-sqlListIntervals` command option first. See also "About SQL Server reports and system tools" on page 384.
`-multiuser`	Sets the server to the `multiuser` run level. For information about the various run levels, see "Server run levels" on page 84.
`-nodefailed` *`serverId`*	Releases any tasks owned by the PolicyCenter server specified by *`serverId`*. Only use this command option if the server referenced by *`serverId`* has already been stopped or otherwise shutdown. See also the `-evenifincluster` option. **IMPORTANT** You must ensure that the server referenced by *`serverId`* is actually stopped if using the `-evenifincluster` option. PolicyCenter does not prevent you from using this option if the server is still running. However, this option overrides an important safety check on the server. It can produce unexpected and possibly negative results if the server is running. Use the `-evenifincluster` option only if both of the following are true: • The server in question is no longer running. • The standard operation of the `-nodefailed` command failed due to the server retaining its cluster membership.
`-nodes`	Provides information about each PolicyCenter server in the cluster. The report contains the following information on each cluster member: • ID of this cluster server • Whether the server instance is actively in the cluster • Server run level • Time the server instance started • Time at which PolicyCenter last updated the server instance • Number of user sessions active on the server instance • Whether a planned shutdown is in progress • Time of the planned shutdown • Whether background tasks are still active on the server The report information is similar, but not identical, to the cluster information available from the Server Tools **Cluster Members** screen. See "The Cluster Members screen" on page 391 for information on that screen.
`-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` option with these available beginning and ending snapshot IDs.
`-oraPerfReport` *`beginSnapshotID endSnapshotID probeVDollarTables`*	Generates a Guidewire AWR performance report using the `OraAWRReport` batch process. This command option has the following arguments: • *`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.

Option	Description
	The third argument can also specify a file by prefixing the file name with an `@` sign, for example, `@`*`filename`*`.properties`. Optionally, you can prefix the file name with the path to the file, if the file is 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`) You must spell and capitalize each property as shown or PolicyCenter ignores the property. If you specify a property, you must set value of that property to either `true` or `false`. If you do not specify a property, PolicyCenter uses the default value for that property. The `-oraPerfReport` option 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 -processstatus` *`processID`* View the performance report on the Info Page. See “The Oracle AWR screen” on page 379.
`-password` *`password`*	Password (*`password`*) to use to connect to the server. PolicyCenter requires the 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” on page 83.
`-recalcchecksums`	Recalculates file checksums that PolicyCenter uses for clustered configuration verification.
`-reloadloggingconfig`	Directs the server to reload the logging configuration file.
`-requestcomponenttransfer` *`type componentId targetOwner`*	Requests transfer of ownership of a component of the specified type (*`type`*) and ID (*`componentId`*) to the specified PolicyCenter server (*`targetOwner`*). Use the `-components` command option to determine the component information to enter. The `-requestcomponenttransfer` command option fails if the component cannot be successfully stopped or the current owning server is unable to process the request.

`-scheduleshutdown` *`serverId`*
`[-terminatebatchprocesses`
`-shutdowndelay` *`minutes`*`]`
	Schedules the planned shutdown of the server specified by *`serverId`*. Use with the following optional options:
	• `-terminatebatchprocesses` – Determines how the shutdown process handles the batch processes (including work queue writers) currently running on the server. See “About planned server shutdowns” on page 392 for information on this command option.
	• `-shutdowndelay` *`minutes`* – Server shuts down in the number of minutes specified by *`minutes`*.

Option	Description
	It is also possible to schedule a server shutdown in the following ways: • From the Server Tools **Cluster Members** screen. See "The Cluster Members screen" on page 391 for information. • Through the `SystemToolsAPI` web service. See the PolicyCenter *Integration Guide* for information.
`-shutdowndelay` *`minutes`*	Server shuts down in the number of minutes specified by *`minutes`*. Use with the `-scheduleshutdown` option only.
`-server` *`url`*	Specifies the PolicyCenter host server URL. Include the port number and web application name, for example: `http://`*`serverName`*`:8180/pc`
`-sessioninfo`	Returns the session information of the server.
`-sqlListIntervals` *`numIntervals`*	Lists the *`numIntervals`* number of most recent Query Store runtime statistics intervals, including its ID. Each interval ID specifies a unique time interval.
`-startfullupgrade`	Starts the process of full application upgrade for the entire cluster of server members. You must shut down all cluster members and deploy a new WAR/EAR file to each one before attempting the upgrade. See the PolicyCenter upgrade documentation for details of the PolicyCenter application upgrade process. See the *Upgrade Guide* for more information.
`-terminatebatchprocesses`	Immediately terminates all running batch processes on the specified server. Use with the `-scheduleshutdown` option only.
`-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 update`*	Launches the Update Statistics batch process to update database statistics. It is possible to specify an optional text description (*`description`*) for this batch process execution. PolicyCenter shows the text of the description on the **Execution History** tab of the **Database Statistics** info page. Specify one of the following values for *`update`*: • `true` – (Default) Update database statistics for tables exceeding the change threshold only. Guidewire defines this change threshold through the `incrementalupdatethresholdpercent` attribute of the `<databasestatistics>` element in `database-config.xml`. • `false` – Generate full database statistics. **IMPORTANT** 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 issues. **See also** • "Understanding database statistics" on page 285 • "Database Statistics work queue" on page 138 • "The Database Statistics screen" on page 375
`-user` *`user`*	User (*`user`*) to use to run this process.
`-verifyconfig` *`filepath`*	Compares the local server configuration with the configuration of the remote cluster. The *`filepath`* parameter defines the path to the local WAR/EAR file that contains the PolicyCenter application, `pc.war`, for example. The command provides an on-screen report that contains information about the feasibility of a configuration upgrade for the server instances in the cluster. For example, the tool provides the following types of information: • Configurations are different – Requires a full PolicyCenter server upgrade. • Configurations are identical – No upgrade is necessary. • Configurations are compatible – Guidewire permits a rolling upgrade.

Option	Description
	If a rolling update is not possible, the command lists the incompatible or missing files. If a rolling update is in progress, there are two possible configurations active in the cluster. Each individual server instance is using either the source configuration or the target configuration. The `-verifyconfig` command option checks for both configurations on the server instances on which you run the command and reports which of the configurations is active on this cluster member. If neither configuration is active, the command reports that a rolling update is in progress and that it is not possible to verify the configuration at this time. See "PolicyCenter system tools" on page 79 for information on how the `-verifyconfig` option works with the substitution of placeholders in the local configuration files.
`-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] {
             -clearerror |
             -clearexclusion |
             -clearstaging |
             -deleteexcluded [-batch] |
             -encryptstagingtbls [-batch] |
             -getLoadHistoryReport reportID [-filepath filepath] |
             -integritycheck [-allreferencesallowed] [-batch] [-clearerror]
                   [-numthreadsintegritychecking num ] [-populateexclusion] |
             -integritycheckandload [-allreferencesallowed] [-batch] [-clearerror]
                   [-estimateorastats] [-numthreadsintegritychecking num ]
                   [-populateexclusion] [-zonedataonly] |
             -listLoadHistoryReports numReports |
             -populateexclusion [-batch] |
             -updatedatebasestatistics [-batch] [-integritycheckandload] }
```

The `table_import` command loads data from staging tables into PolicyCenter. Most of the options for this command require the server to be at the `MAINTENANCE` run level. Before you use those command options, use the `system_tools -maintenance` command option to set the server run level to `MAINTENANCE`. Use the `system_tools -multiuser` command option to set the server run level to `MULTIUSER` after the table import command completes.

It is not possible to use the `system_tools -terminateprocess` command option to terminate the `table_import` command.

IMPORTANT PolicyCenter supports bulk data import from staging tables for loading zone data only. For more information, see "zone_import command" on page 439.

See also

- "The Load History Information screen" on page 386
- "system_tools command" on page 429
- *Integration Guide*

table_import command options

You can use any of the following options with the `table_import` command. You must always supply the `-password` option.

Option	Description

`-allreferencesallowed`	Allows references to existing non-administrative rows in all operational tables.
	This option only applies with the following command options:
	`-integritycheck`
	`-integritycheckandload`
	This option corresponds to the Boolean parameter `allowRefsToExistingNonAdminRows` used by the integrity check methods of the `TableImportAPI` web service. Guidewire recommends that you use this option or the equivalent API parameter, set to `true` only if absolutely necessary.
	This option can cause performance degradation during the check and load process, which would noticeably slow down the loading of staging table.
	See also "The load history detail report" on page 387.

`-batch`	Runs the `table_import` command in a batch process. This option only applies with the following command options:
	`-deleteexcluded`
	`-encryptstagingtbls`
	`-integritycheck`
	`-integritycheckandload`
	`-populateexclusion`
	`-updatedatabasestatistics`
	You can run `table_import` in a batch process from any node in a PolicyCenter cluster. However, table import batch processing must run physically on a server designated as a `batch` server. Therefore, in running the command, provide the URL of a batch server and also provide the user credentials for that batch server.
	Note: The `-batch` option does not wait until the started process completes before returning. Instead, it returns immediately and prints the ID of the started process (PID). The process caller is responsible for waiting for the process to complete before taking further action.

`-clearerror`	Clears the error table. See also "The load history detail report" on page 387.
`-clearexclusion`	Clears the exclusion table.
`-clearstaging`	Clears the staging tables. Requires the server to be at the `MAINTENANCE` run level.
`-deleteexcluded`	Deletes rows from staging tables based on contents of exclusion table.
`-encryptstagingtbls`	Do not use. While this command option is available, Guidewire does not support this command option in PolicyCenter.
`-estimateorastats`	Executes queries for row counts on production tables and sets the database statistics. If you do not use this option, the import command uses information in database statistics to report approximate row counts. Use the `-estimateorastats` option only to load production tables that are empty or have very few rows. Used with the `-integritycheckandload` command option. This option only applies with the following command option: • `-integritycheck` • `-integritycheckandload` This parameter corresponds to the Boolean parameter `updateDBStatisticsWithEstimates` used by the integrity check methods of the `TableImportAPI` web service. This command option applies to Oracle databases only.
`-filepath` *`filepath`*	Path to target directory in which to download a report. Use with the `-getLoadHistoryReport` command option.
`-getLoadHistoryReport` *`reportID`*	Downloads a compressed Zip version of the load history report as specified by the value of *`reportID`*. Does not require the server to be at the `MAINTENANCE` run level. Use the `-listLoadHistoryReports` option to determine the ID to use. Use the optional `-filepath` parameter to specify the target directory for the download.

Option	Description

`-integritycheck`	Validates the contents of the staging tables. You can optionally specify:
	`-allreferencesallowed`
	`-clearerror`
	`-numthreadsintegritychecking`
	`-populateexclusion`

`-integritycheckandload`	Validates the contents of the staging tables and populate operational tables. You can optionally specify one of the following command options as well:
	`-allreferencesallowed`
	`-clearerror`
	`-estimateorastats`
	`-numthreadsintegritychecking`
	`-populateexclusion`
	`-zonedataonly`

`-listLoadHistoryReports [`*`numReports`*`]`	Lists the most recent load history reports. Optional parameter *`numReports`* is the number of reports to list: • If you supply a positive integer for *`numReports`*, then PolicyCenter lists that number of most recent reports. • If you do not supply a value for *`numReports`*, then PolicyCenter lists all available reports. Does not require the server to be at the `MAINTENANCE` run level.
`-messagesinks` *`sinks`*`, ...`	Deprecated. This option does nothing.

`-numthreadsintegritychecking` *`num`*	Specifies the number of threads that PolicyCenter is to use in running database table integrity checks. The value of *`num`* has the following meaning:
	• Not specified – PolicyCenter assumes the number of threads to be one, no multithreading.
	• 1 – No multithreading, the default.
	• 2 - 100 – PolicyCenter runs the database integrity checks with the number of specified threads.
	• > 100 – PolicyCenter throws an exception.
	This option only applies with the following command options:
	`-integritycheck`
	`-integritycheckandload`

`-password` *`password`*	Password (*`password`*) to use to connect to the server. PolicyCenter requires the password.
`-populateexclusion`	Populate the exclusion table with rows to exclude. See "The load history detail report" on page 387.
`-server` *`url`*	Specifies a PolicyCenter server URL. Include the port number and web application name, for example: `http://`*`serverName`*`:8180/pc` If running the table import command in a batch process, see `-batch` for more information.
`-updatedatabasestatistics`	Updates the database statistics on the staging tables. Run the table import command with this option after populating the staging tables, but before using the `-integritycheck` or `-integritycheckandload` options. See "The load history detail report" on page 387.
`-user` *`user`*	Specifies the user to use to run this process.
`-zonedataonly`	Sets the import to load zone data only. Used with the `-integritycheckandload` command option.

template_tools command

```
template_tools -help
template_tools -password password [-server url] [-user user] {
               -convert_dir directory |
               -convert_file filename [working_dir directory] |
               -import_dir objectsfile fieldsfile directory [working_dir directory] |
               -import_files objectsfile fieldsfile outfile |
               -list_templates |
               -validate_all |
                -validate_template templateID }
```

The `template_tools` command contains options to list, manage, and validate document templates.

See also

- *Integration Guide*

template_tools command options

You can use any of the following options with the `template_tools` command. You must always supply the -`password` option.

Option	Description
`-convert_dir` *`directory`*	Converts all templates in the specified directory to the new format.
`-convert_file` *`filename`*	Converts the specified template to the new format.
`-import_dir` *`objectsfile fieldsfile directorys`*	Imports context objects and form fields from the provided CSV-formatted files into all the templates in the specified *`directory`*. This option has the following arguments: • *`objectsfile`* – File containing the context objects for import, in CSV format. • *`fieldsfile`* – File containing the fields for import, in CSV format. • *`directory`*– Directory that contains the templates to update.
`-import_files` *`objectsfile fieldsfile outfile`*	Imports context objects and form fields from the provided CSV-formatted files into the specified template descriptor file (*`outfile`*). This option has the following arguments: • *`objectsfile`* – File containing the context objects for import, in CSV format. • *`fieldsfile`* – File containing the fields for import, in CSV format. • *`outfile`* – Template descriptor file to update.
`-list_templates`	Lists all of the templates available for validation.
`-password` *`password`*	Password (*`password`*) to use to connect to the server. PolicyCenter requires the password.
`-server` *`url`*	Specifies the PolicyCenter host server URL. Include the port number and web application name, for example: `http://`*`serverName`*`:8180/pc`
`-user` *`user`*	User (*`user`*) to use to run this process.
`-validate_all`	Validates all the templates in a similar manner to `-validate_template`.
`-validate_template` *`templateID`*	Validates a single template. Validates that the given template descriptor (*`templateID`*) is in a valid format, and that all template descriptor context objects and form fields are valid given the current data model.
`-working_dir` *`directory`*	Specifies a directory for use as the root (working directory) for relative paths.

workflow_tools command

```
workflow_tools -help
workflow_tools -password password [-server url] [-user user] {
               -complete workflowID |
               -resume workflowID |
               -resume_all |
               -suspend workflowID }
```

You can also control workflows using the `WorkflowAPI` web service. See the *Integration Guide*.

workflow_tools command options

You can use any of the following options with the `workflow_tools` command. You must always supply the `-password` option.

Option	Description
`-complete workflowID`	Completes running workflow for the specified workflow (*workflowID*).
`-password password`	Password (*password*) to use to connect to the server. PolicyCenter requires the password.
`-resume workflowID`	Resume named workflow (*workflowID*) in the error or suspended state.
`-resume_all`	Resume all workflows in the error or suspended state.
`-server url`	Specifies the PolicyCenter host server URL. Include the port number and web application name, for example: `http://serverName:8180/pc`
`-suspend workflowID`	Suspend the named workflow (*workflowID*).
`-user user`	User (*user*) to use to run this process.

zone_import command

```
zone_import -help
zone_import -password password [-server url] [-user user] {
            -import filename -country country [-clearstaging] [-charset charset] |
            -clearproduction [-country country] |
            -clearstaging [-country country] }
```

The `zone_import` command imports data in CSV format from specified files into database staging tables for zone data. It is only possible to import zone data for a single country at a time. The zone data files that you import must contain zone data for a single country only. To load zone data for multiple countries, use the command multiple times with different, country-specific zone data files each time.
Guidewire expects that you import address zone data upon first installing PolicyCenter, and then at infrequent intervals thereafter as you receive data updates.

See also

- For a discussion of zone data, importing a zone data file, and working with custom zone data files, see “About importing zone data” on page 313.
- For more information on importing zone data and database staging tables generally, see the *Integration Guide*.
- For information on the web service `ZoneImportAPI` that also imports zone data, see the *Integration Guide*.

zone_import command 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` option to clear data for a single country only.
`-clearstaging`	Clears zone data from the staging tables. Optionally, specify the `-country` option to clear data for a single country only.
`-country` *`countrycode`*	Used with `-import`, `-clearproduction`, and `-clearstaging` command options: • If used with the `-import` option, `-country` specifies the country of the zone data in the import file. • If used with either the `-clearproduction` or `-clearstaging` options, `-country` specifies the country of the zone data to clear from the tables.
`-import` *`filename`*	Imports zone data from the specified file (*`filename`*). You must set a value for the `-country` option. If you include the optional `-clearstaging` option, PolicyCenter clears the data in the staging tables for the specified country before importing the data from the import file.
`-password` *`password`*	Password (*`password`*) to use to connect to the server. PolicyCenter requires the password.
`-server` *`url`*	Specifies the PolicyCenter host server URL. Include the port number and web application name, for example: `http://`*`serverName`*`:8180/pc`
`-user` *`user`*	User (*`user`*) to use to run this process.